TASK: UU03
CDRL: 05156
31 July 1993

Organization Domain Modeling Volume I - Conceptual Foundations, Process And Workproduct Description

Informal Technical Data

STARS-UC-05156/024/00
31 July 1993

TASK: UU03
CDRL: 05156
July 31, 1993

INFORMAL TECHNICAL REPORT

For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Organization Domain Modeling (ODM)
Volume I - Conceptual Foundations, Process and Workproduct Descriptions
Version 0.5—DRAFT

STARS-UC-05156/024/00
July 31, 1993

Data Type: A005, Informal Technical Data

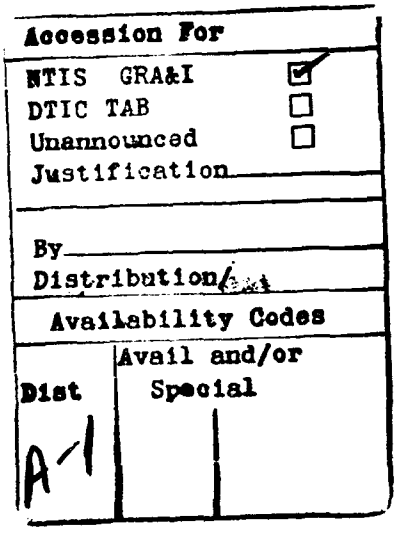
CONTRACT NO.: F19628-88-D-0031
Delivery Order 0011

Prepared for:

Electronic Systems Center
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by

Paramax Systems Corporation
Sunrise Valley Drive
Reston, VA 22091



Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

TASK: UU03
CDRL: 05156
July 31, 1993

Data ID: STARS-UC-05156/024/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: UU03
CDRL: 05156
July 31, 1993

INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM)
Volume I - Conceptual Foundations, Process and Workproduct Descriptions
Version 0.5 - DRAFT

Principal Authors:

Mark A. Simos *Date*

Task Manager *Richard E. Creps* *Date*

(Signatures on File)

TASK: UU03
CDRL: 05156
July 31, 1993

INFORMAL TECHNICAL REPORT
Organization Domain Modeling (ODM)
Volume I - Conceptual Foundations and Process Descriptions
Version 0.5 - DRAFT

Change Record:

Data ID	*Description of Change*	*Date*	*Approval*
STARS-UC-05156/024/00	Version 0.5: Draft Issue	July 1993	*on file*

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	31 July 1993	Informal Technical Data

4. TITLE AND SUBTITLE

Orgnization Domain Modeling Volume I - Conceptual Foundations, Process And Workproduct Description

5. FUNDING NUMBERS

F19628-88-D-C031

6. AUTHOR(S)

Mark A. Simos, Richard E. Creps

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER

CDRL NBR
STARS-UC-05156/024/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
ESC/ENS
Hanscom AFB, MA 01731-2816

10. SPONSORING MONITORING AGENCY REPORT NUMBER

05156

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution "C"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Organization Domain Modeling (ODM) is a general method for domain analysis (DA) and modeling, including a structured set of workproducts, a tailorable process model and a set of modeling techniques and guidelines. This technical report describes the conceptual foundations of ODM, and outlines the processes and workproducts defined by the method.

14. SUBJECT TERMS

15. NUMBER OF PAGES
180

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

TABLE OF CONTENTS

Table Of Contents: Appendices

List of Figures

1. Executive Summary

Organization Domain Modeling (ODM) is a general method for domain analysis (DA) and modeling, including a structured set of workproducts, a tailorable process model and a set of modeling techniques and guidelines. This technical report describes the conceptual foundations of ODM, and outlines the processes and workproducts defined by the method.

ODM was first prototyped as part of the design process for the Reuse Library Framework (RLF). Further developed through the author's work as an independent consultant on software reuse, the ODM method has been refined with support from Hewlett-Packard Corporate Engineering's Reuse Program and, more recently, Paramax Corporation as a prime contractor for ARPA's STARS program. The method is currently being tailored for use on various HP and STARS projects, and training materials and tutorial examples are being collaboratively developed.

As part of the STARS-supported work on ODM, ODM processes and workproducts have been structured in accord with the reuse process model of the CFRP (Conceptual Framework for Reuse Processes) [STARS92b]. The CFRP's Reuse Management Cycle (Plan - Enact - Learn) is reflected at several lower levels of the ODM process model, in the structure of ODM workproducts, and in ODM strategies for technology transfer, training, and ongoing evolution. ODM processes refine CFRP process categories concerned with modeling domain-specific information for reuse. In CFRP terms, the scope of ODM is primarily that of a single Asset Creation (i.e., domain engineering) project. ODM's domain planning and learning activities directly reflect CFRP process categories. The ODM method is based on a critical distinction between descriptive and prescriptive modeling. These modeling activities are orthogonal to the Domain Analysis and Modeling, and Software Architecture Development phases within the CFRP Asset Creation process family. These two complementary perspectives on the domain modeling process can be integrated to generate a number of specific planning approaches and process models for domain engineering projects.

As with the CFRP in general, ODM processes and workproducts are tailorable to a variety of organizational contexts. An explicit goal in developing ODM was to produce a process model and method applicable across government, contractor and commercial software organizations. ODM can serve as a framework for comparing results from a number of different DA methods and representations, both formal and informal. For example, HP researchers have used the ODM process model to structure design reviews of existing application projects that have conducted informal domain analyses, (or processes resembling domain analyses, such as the development of reusable architectures) as part of their mandated design process.

STARS is currently supporting integration of the ODM method more closely with RLF as a domain modeling tool. While not dependent on RLF, ODM motivates use of inheritance-based modeling techniques like the RLF's semantic network representation scheme for domain modeling, by proposing an approach to systematic use of specialization semantics in domain modeling. Collections of reusable assets (or asset bases) developed and structured in this way allow users to make strong assumptions about the behavior of reusable assets,based on their location within an inheritance network. This aids in both qualification and retrieval of reusable assets.

Other distinctive features of ODM include the following:

1) A rich set of inter-domain relation types help clarify domain boundaries and interfaces during domain scoping.

2) An explicit representative set of system *exemplars* are selected for the domain that provide an empirical basis for modeled features and decisions about the intended scope of applicability for assets.

3) Models of system features are augmented with historical and developmental process information about the domain, drawing on and adapting relevant methods from work in knowledge acquisition and workplace ethnography.

4) A range of informal to formal (e.g., inheritance-based) techniques are employed to produce descriptive domain models of requirements-oriented features; these domain models can be linked to test cases and scenarios providing operational semantic descriptions for model elements.

5) A domain architectural model is derived that captures prioritized system variants in layered and/or separately selectable configurations.

6) A final technology selection phase addresses component-based, generative, and hybrid encapsulation techniques for reusable assets.

2. Introduction

This technical report introduces **Organization *Domain Modeling*** (ODM), a formalization of an approach to domain analysis (DA) for software reuse developed by the author over the past several years. This report outlines the conceptual foundations, major processes, constituent models and other workproducts of ODM, and discusses experience with the method to date. This introduction outlines the purpose, scope and organization of the report, and the assumed background of the reader, and acknowledges fellow contributors to the ODM method.

2.1 Document Scope and Status

The ODM method is currently evolving rapidly through the work and experience of early adopters (see Section 3.2, Progress to Date). Its main focus is to present the ODM process model and workproduct set, along with some guidelines and heuristics for performing each step. The scope of the ODM process is described later in this document, in Section 4.1. The document does not as yet constitute a full training manual or practitioner's guide to ODM. The report will be most useful as documentation to an ODM training session, to be accompanied by periodic follow-up support.

The genesis of this document was a short technical paper providing a high-level summary of the ODM process. As practitioners on the STARS/Army Demonstration Project began applying the method on a full-scale project, the need for detailed process descriptions and explanation of concepts became critical, and this report expanded in an attempt to capture ongoing lessons learned and emerging insights as rapidly and thoroughly as possible. The document in its present form therefore represents a kind of snapshot of the current state of

the method. This approach has led to a great amount of new technical content in the report, at the expense of a thorough editing and review cycle.

Readers should be aware of at least the following known problems with the current form of the document, in order to use it to best effect:

1) Terminology used for ODM processes, workproducts, and key concepts is still in evolution. Some terms may be inadequately defined, especially in light of their connotations in other contexts familiar to some readers. Even apparently innocuous terms such as *workproduct* and *artifact* are used with particular senses in the ODM context; the distinctions involved are somewhat subtle and are still being resolved in discussions and practice. There may be some inconsistency in terms in different parts of the document that have yet to be resolved. Hopefully, the overall complexity of the terms used can be reduced once the key distinctions are more fully matured.

2) The current document lacks formal process diagrams, such as SADT-style notations to describe the ODM process. Such diagrams are key components of the documentation for several DA methods such as the DAPM [Prie91] and DISA [DISA93] methods. Future versions of this report or additional ODM documents will contain these elements, once better validated in use.

The ODM method does present particular challenges in adapting process modeling notations. ODM is intended to be applicable in a wide variety of organizational contexts. Specific processses and workproducts selected, and the sequence in which processes are performed may vary widely across these contexts; it is difficult to express this degree of variability using conventional process modeling techniques. Future versions of this technical report may therefore include a set of process diagrams from a particular project's *instantiation* of the ODM process model. Readers intending to apply the ODM method would not follow these diagrams literally, but instead would use them as examples in generating their own ODM process plans and models.

3) The current document does not include formats for the various workproducts. The emphasis in the ODM method has been on defining the required **content** of various models and other workproducts, and leaving to the discretion of the project team the particular textual and/or graphic notations to employ, along with appropriate automated tools. However, it would clearly be helpful for readers to have templates and/or examples of the various workproducts for understanding, if not for direct adaptation.

4) A thorough, running example domain is needed to clearly explain all the various processes and workproducts in ODM. Working through a small-scale domain example following the detailed process in this document was not possible given the resources available for this version of the report. Once again, future versions of this document or additional ODM documents should include detailed examples drawn from a realistic DA project.

5) For most readers, a greater use of graphic illustrations would help clarify the concepts in this document. A selection of some illustrations has been included in this version, drawn from a more extensive set of viewgraphs used in ODM training sessions. Not all these diagrams are in complete agreement with the text, and some essential illustrations are not yet available for inclusion.

6) Finally, readers will no doubt find numerous conceptual and formatting inconsistencies, omissions, and repetitions in this document which await a more extensive review and editing cycle. It is hoped that the value of getting ODM concepts out into the reuse researchers' and practitioners' community for experimentation, review and critique will outweigh the roughness of the exposition.

2.2 Intended Audience

This technical report provides prescriptive guidance to domain analysts and domain engineers, process engineers, and project managers who have been or expect to be involved in a DA project. The report assumes an audience with a background in software engineering from a technical and/or managerial perspective. It assumes the reader is acquainted with work in software reuse and, if not a zealot, is at least willing to suspend disbelief concerning the potential benefits of reuse. Arguing the merits and feasibility of reuse in general is beyond the scope of this report.

The presentation of ODM in this document depends heavily on concepts from the Conceptual Framework for Reuse Processes [STARS92b], a joint STARS document that proposes a set of fundamental reuse process categories and concepts. Much of the rationale for the organization of the ODM process model, details of activities, and terminology used in the report will be apparent only with some knowledge of the CFRP. At a minimum, readers should be familiar with the introductory paper on the CFRP [Cre92], preferably with the definition and application documents [STARS92b, STARS93].

While some readers will undoubtedly be familiar with other DA methods, either through reading or practice, the document can be read without this background. For those curious about how ODM compares with these methods, detailed comparisons with some well-known methods are included in an appendix. Readers familiar with other DA approaches are cautioned that ODM defines certain terms (e.g., exemplar set of systems) in ways that differ subtly but significantly from the terms' usage in other approaches.

2.3 Document Organization

This document is organized as follows:

- SECTION 1: Executive Summary.
- SECTION 2: Introduction—introduces this report, describing the scope, intended audience, and document organization.
- SECTION 3: ODM Overview—provides an overview of ODM, including background on the genealogy of the method, its motivation and objectives.
- SECTION 4: ODM Context—describes the intended scope of ODM and its relationship to other work in the area of reuse processes.
- SECTION 5: ODM Process Overview—presents a condensed description of the full ODM process, from descriptive through prescriptive and implementation modeling.

- SECTION 6: ODM Principles—Outlines some differentiating features of ODM, and presents a model of core principles that underlie and integrate all ODM processes and workproducts.

- SECTION 7: ODM Processes and Workproducts—This core section of the document presents the ODM process model in detail, with associated models and other workproducts developed at each stage of the process, and some discussion of guidelines and issues to be considered at various phases.

- SECTION 8: Comparison with Other Methods—provides a brief comparison of the ODM method with some other prominent DA methods.

- SECTION 9: Future Work—discusses near-, mid-, and long-term plans for the future evolution of the ODM method.

- SECTION 10: Conclusion.

- APPENDIX A: Guidelines for Domain Identification—provides a more in-depth discussion of the diverse organizational scenarios in which a domain analysis effort might be initiated. This is an example of an extended level of discussion that might be suitable for an ODM practitioner's guidebook.

- APPENDIX B: Lexicon Specification Language Definition and Example—a sample BNF for an ODM workproduct format and a small example of such a workproduct. Future versions of ODM might include a set of such mini-specification languages.

- APPENDIX C: References.

- APPENDIX D: ODM Lexicon—A preliminary collection of terms and definitions for the ODM method.

- Index—also a preliminary version of a cross-reference for terms used within the document.

2.4 Examples

The author has created an initial set of example models for the domain of text outlining programs as baseline material for an internal Hewlett Packard tutorial on DA. Some examples given in this document will be drawn from this domain. Other examples may be drawn from the STARS Paramax/Army CECOM Demonstration Project.

2.5 Disclaimer

Though this technical report was funded by the STARS program, the report and the ODM method in general reflect the author's individual work in this area. The intent of the document is to make the concepts accessible to more thorough review and evolution by the broader reuse research community. ODM concepts, while generally compatible with and supportive of the STARS approach to reuse, are the sole responsibility of the author, and are not necessarily endorsed by STARS in the form presented here. Similarly, though much material presented in this document was refined with the help of researchers at Hewlett

Packard, the method itself is neither proprietary to nor endorsed by HP, nor tailored, in the form presented here, to an HP-specific context.

2.6 Acknowledgments

The author wishes to thank some of the many colleagues that have helped to refine the basic concepts presented in this report: the original RLF team at Unisys' Paoli Research Center; the STARS Reuse CFRP Team members; Hewlett Packard's Software Reuse group (within Corporate Engineering's Software Initiative); tolerant early adopters in HP engineering divisions; members of the HP Labs Software Reuse Department; colleagues and fellow researchers in reuse that have helped to develop and critique these ideas over the course of yearly reuse workshops since 1986; and participants from the STARS TT Affiliates Program, and the CARDS and PRISM programs, who contributed useful feedback on ODM at preliminary training sessions. Patricia Collins (HP Corporate Engineering) and Richard Creps, Paramax STARS Reuse Chief Engineer, were both instrumental in supporting the development of ODM and contributed greatly to its technical content. The STARS/Army Demonstration project team, under the project management of John Willison, and the Domain Engineering team, with Maj. Grant Wickman as technical lead, are making significant contributions to the method through their efforts. The author also would like to acknowledge as one of ODM's intellectual forebears Dr. Michael Freeman, progenitor of the KNET system, a Prolog-based KL-ONE -style knowledge representation system on which the original RLF was based [KNET???].

3. ODM Overview

This section provides an overview of the motivations for ODM with respect to other work in software reuse and domain analysis; historical background on the development of the method; progress to date; and high-level objectives for the method.

3.1 ODM Motivation

Reuse researchers, planners and advocates in government and industry have come to view domain analysis (DA) as an essential element of a systematic reuse program. Growing interest is evidenced by new tools claiming to support DA [Bail92, MC89], recently published DA approaches, methods and case studies [Prie91, Kang90], and an emerging secondary literature of comparisons, surveys and even "domain analyses" of DA methods [Aran91, Wartik92]. While no method has yet been accepted as a standard, there appears to be an emerging consensus on a core set of concepts relevant to DA.

Despite this burgeoning activity, there remain theoretical and practical challenges to formalizing DA as a well-understood software engineering discipline. Beneath commonly accepted terms—e.g., horizontal vs. vertical domains, domain architectures, etc.—lurks an undercurrent of conceptual fuzziness and persistent uncertainty about just what DA is (and isn't). One reason for this instability of terminology is the relative immaturity of the discipline [Kuhn???]. Another factor is the hybrid nature of the discipline. DA has adopted concepts and techniques from fields as diverse as library information science, knowledge representation, object-oriented analysis and design, and market analysis, among others, as suggested in Figure A. As a consequence, many terms such as "domain", "architecture", and "inheritance" have overloaded meanings for the various communities involved with research in DA.

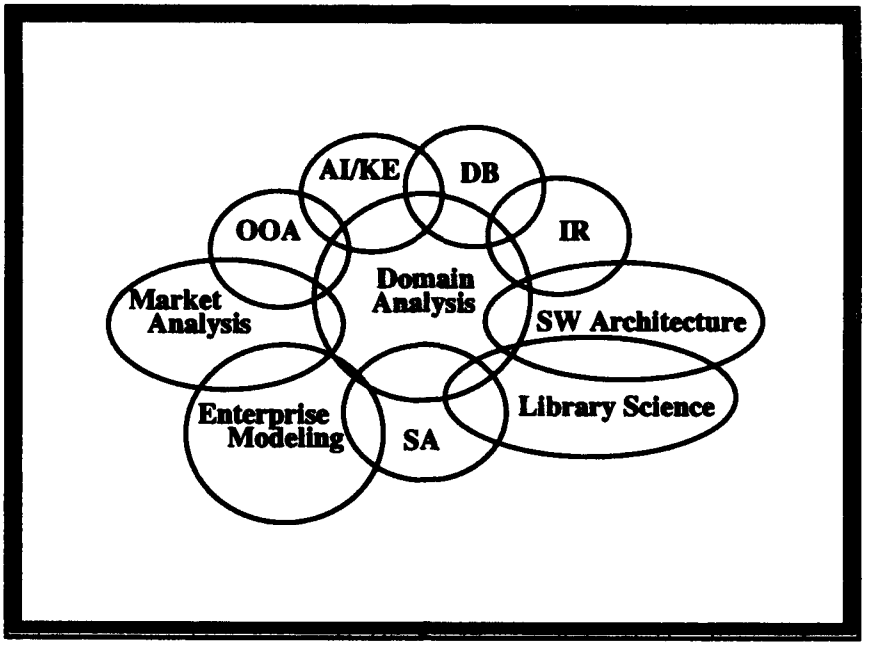


Figure A: DA as a Hybrid of Other Disciplines

The upsurge in interest in DA, combined with the lack of formal foundations, has led in some cases to hasty adoption of what are labeled domain analysis processes, without clear explanation of how these processes differ from traditional software analysis techniques. This has done little to allay skepticism about DA. For example, many engineers doubt that components engineered for reuse can provide adequate optimization for applications with stringent performance requirements. These fears are exacerbated by claims that a domain model should include only high-level requirements, ignoring commonality at levels of concern to designers and implementors. Similarly, when engineers are shown examples of DAs that appear to be merely generic system designs, they are right to question why the supposed domain analyst's design decisions carry more weight than those of application specialists. Proposals for institutionalizing reuse sometimes assume that use of assets can be mandated by policies and incentives to overcome "resistance", when this resistance may be in part a justified response to fundamental design and engineering issues that have not been satisfactorily addressed.

Conversely, in adopting too narrow a view of DA, many engineering projects have been ignored that, while perhaps not initiated as domain analyses, share key features distinguishing them from conventional system engineering efforts in a way strongly analogous to DA. If we had a methodical way of identifying such informal processes as variants of DA they could provide valuable empirical examples for comparing DA methods, terminology and practical experience. One strong motivation for the formalization of the ODM method, therefore, is the perceived need for clear criteria of what constitutes DA processes.

3.2 ODM Genealogy and Progress to Date

The ODM approach derives in part from a process prototyped by the original design team for the Reusability Library Framework (RLF), a hybrid semantic network (AdaKNET) and rule-based (AdaTAU) knowledge representation system. While derived in part from modeling approaches used in the expert systems field, RLF was intended for use as a domain modeling tool [Sim88,Wall88,MC89]. The author was the initial technical lead on the project. Layered architectures and phased implementation plans for both the semantic network and rule-based subsystems of the RLF were developed by performing a feature analysis on a small set of exemplar systems [Unisys88]. RLF was in fact intended for use as a support tool for a fuller version of this modeling method. However, the results of the RLF domain analysis were never themselves represented in RLF workproducts.

The ODM approach has subsequently been evolved by the author as an independent consultant in the field of reuse. Technical aspects of the approach have been integrated with broader organizational management perspectives [Sim91a, Sen90]. The method has been refined through presentations and talks to industry and university researchers [Sim90a, Sim90b], small-scale trial applications in some example domains (font management systems, computer music sound samples [Sim91b], text outliners), and training sessions delivered to research and engineering groups within Hewlett Packard, the STARS and CARDS program personnel. Several DA training packages reflecting aspects of the ODM approach are currently in preparation. In collaboration with researchers at Hewlett Packard, the author has drafted a reuse process model, question sets and guidelines incorporating much of the approach to domain modeling outlined here.

Some aspects of the ODM approach have been presented to HP application teams through the medium of RADAR (Reusability Analysis/Domain Analysis Review) sessions [Col92]. These are design reviews focusing on projects that have followed a system design or architectural modeling process similar to DA, whether or not a formal DA method was followed. Through questions about their own design process, engineers learn about methods of domain modeling, and gaps are revealed in both the design process followed and in the formal process model itself. These review sessions have revealed the value of various ODM models as instruments for capturing and describing domain knowledge. As one participant put it, "...we didn't know how much we knew."

The best of theoreticians awaits the practitioner with bated breath. ODM is currently being utilized in a full-scale domain analysis effort by the STARS Army/Paramax Demonstration Project at CECOM SED, Ft. Monmouth, NJ. The current form of ODM is truly being co-evolved with the participation of this team. This technical report represents a snapshot of many of the recent insights and refinements to the method that have emerged from these training sessions and trial applications.

3.3 ODM Objectives

ODM synthesizes applicable elements (specific processes, modeling techniques, and suggested workproducts) from a number of other DA methods. At the same time, ODM represents an attempt to address some gaps in current approaches to DA. Most existing DA processes embed contextual assumptions of all kinds: about the overall business contexts in which DA will be performed, characteristics of certain kinds of domains, or the reuse technology (component library, application generator, etc.) that will be used to implement

domain assets [Wartik92]. Such methods do not transfer easily across organizations, particularly to commercial software domains. ODM represents an attempt to preserve *neutrality* in various respects with regard to the domain analysis process. The strategy for achieving such neutrality is threefold: 1) develop a typology for the variants to be supported; 2) make characterization of the current context an explicit step in the process model, using the typology as a resource; 3) provide guidelines and heuristics for tailoring variant elements of the method based on this contextual characterization. The following paragraphs outline some dimensions of this neutrality principle relevant for ODM.

Organization Neutrality

ODM is intended to be tailorable to a variety of organizational contexts, from small- to medium-scale software engineering organizations specializing in one or a few strategic domains to large-scale software organizations spanning many lines of business, in both government and private sector software development contexts. This renders the ODM process model more complex at first glance than a model suitable for a single organization context of use, since in many cases sequences of activities can vary considerably across different organizational scenarios. ODM devotes comparatively more attention to "upstream" domain planning activities such as organization assessment and context-setting, domain selection, definition and scoping. Where possible, the method includes guidelines and heuristics for configuring specific instantiations of the ODM process model for particular projects; as experience with ODM increases, this strategic model and tailoring guidance will be refined considerably.

Technology Neutrality

Many existing DA methods embed assumptions about the technology to be used in implementation (i.e., a library of reusable software components, a language processor, a decision model, etc.) These assumptions affect the choice of domains, the information considered relevant to the model, and even the definition of who the "domain experts" are and their eventual role relative to the technology.

The ODM method does not require use of a particular toolset, or a particular technical approach to asset implementation. On the contrary, ODM workproducts constitute a framework for making the selection of technology, based on organization and domain characteristics and available technology. ODM emphasizes tightly scoped, interconnected domains, and use of a taxonomic modeling representation that is not directly descriptive of code components. This modular approach to domain scoping aids in developing technology-independent architectural frameworks and asset specifications that facilitate the mixture of component- and generator-based implementations on even an asset-specific basis.

Domain Neutrality

ODM does emphasize the definition of relatively small, well-scoped domains. However, the method can be applied to a domain of any size. For larger domains (such as domains covering the scope of entire application systems) the most likely outcome is that a number of relatively complex sub-domains will be identified early in the process and either deferred or modeled as recursive invocations of the ODM process on the smaller scope. It is then

possible to complete the modeling of the larger domain by studying alternative configuration strategies for the subdomains.

Cross-domain (lateral/diagonal) reuse

Most current efforts in reuse are oriented towards fostering reuse **within** domains, i.e., across applications within the same system or product family. Earlier reuse efforts focused on small-scale horizontal domains (e.g., sorting and searching, data structures, user interface functions). While ODM is suitable for either of these types of domains, a specific objective of the method is to foster *cross-domain* reuse, or what might be termed, in distinction to vertical or horizontal, *lateral* or *diagonal* domains. These would include domains with significant structural complexity, allowing for the reuse of relatively large-scale components, but engineered in such a fashion that these sub-systems can be reused within diverse vertical contexts. As an example, the text outlining domain could yield a reusable subsystem to be used within text editors, presentation managers, general on-line help systems and file managers.

3.4 Terminology

In this document, the ODM process model and workproduct set, along with the guidelines associated with these models, will be referenced simply as "ODM". The term "domain modeling" or "modeling" alone will be used almost exclusively in lieu of the more familiar "domain analysis". We interpret domain modeling to be an activity that balances conventional analytic techniques with synthetic or syncretistic thinking (i.e., abstracting general patterns from a set of concrete examples) in performing tasks such as dynamically and iteratively defining domain boundaries. Where the phrase "domain analysis" (or DA) occurs, it refers more broadly to any of the various methods for domain analysis, including ODM, and usually involves a contrast between domain analysis and systems analysis.

The document assumes that domain modeling will be undertaken as a team project, and hence will frequently refer to the ODM team; or the modeling team. This terminology is not meant to exclude cases where an individual performs ODM activities or even an entire project.

Because conventional usage narrowly constrains the interpretation of the term "domain expert", the ODM method uses the term *domain stakeholders* to refer to the various people who, literally, "have a stake" in the domain. Stakeholders include *domain informants*, from whom domain information is gathered, and who may in turn become utilizers of assets produced from the domain modeling project.

The ODM method can present an imposing number of special terms for processes, models and workproducts. In addition, a number of terms are given special meaning within the context of the method. When a term used with special emphasis is first used in the report, it will generally be italicized and briefly defined in line. The ODM Lexicon provided with this report (See Appendix ???) provides further clarification of these terms. Capitalized phrases (e.g., Domain Interconnection Model, Domain Identification) will usually refer to specific workproducts, models or processes within ODM. (The reader is forewarned that terminology in this document is still very much in draft form; many inconsistencies remain in the text.)

4. ODM Context

ODM has evolved in the context of a number of other reuse-related conceptual frameworks, techniques, methods and tools. This section outlines a layered structure within which ODM's role and structure can be better understood in relation to this context.

4.1 ODM Scope

This section describes the intended scope of applicability for ODM, in terms of the processes supported, the commitment to particular representation formalisms, and the extent of explicit support for organizational development processes. The CFRP outlines a set of categories for reuse processes that are intended to span all reuse-specific aspects of reuse-based software development processes. This general framework is partitioned into two idioms, one dealing with Reuse Management (planning, enactment and learning processes), one dealing with Reuse Engineering (creating, managing and utilizing reusable software assets).

The primary focus of ODM is within the Asset Creation process family of the Reuse Engineering idiom. ODM provides a detailed process model for how a software domain is defined and scoped, and descriptively modeled; and how the specific range of functionality to be addressed in the asset base to be developed is derived from further transformations on these models. ODM also encompasses aspects of Reuse Planning and Reuse Learning families within Reuse Management that are directly relevant to domain modeling. This scope is illustrated in Figure B.

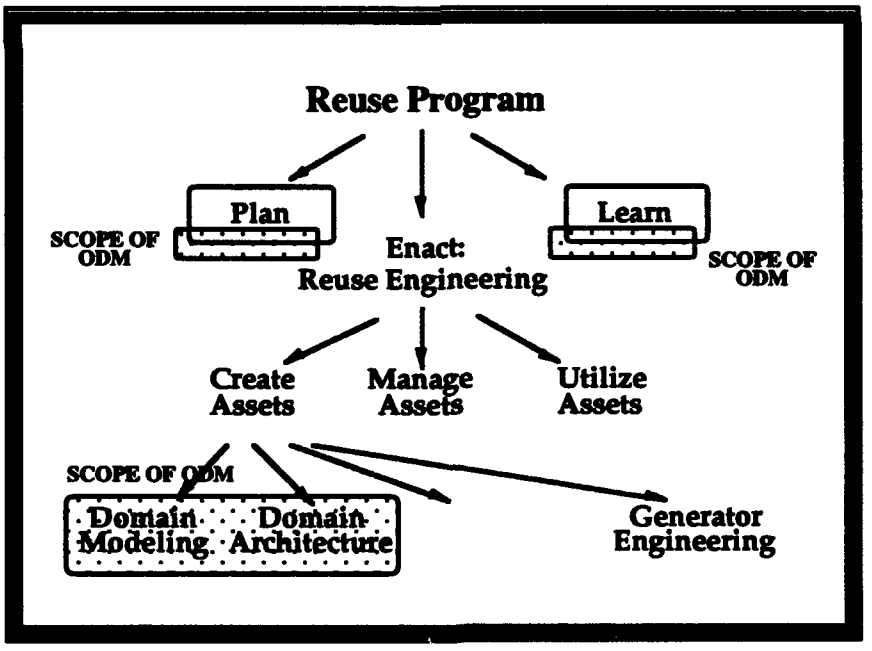


Figure B: Scope of ODM in "Canonic" CFRP Reuse Program

While ODM does encompass a preparatory planning phase for asset implementation that includes selection of implementation techniques, the ODM process sequence concludes at the point that asset specifications are handed off to implementors. The rationale is that the

set of skills required for implementing reusable assets, whether via component-based or generator-based engineering, is more closely related to application engineering skills than to those of domain modeling.

4.2 ODM and the Organon Vision

ODM is grounded in a new approach to understanding the role of domain-specific knowledge within organizational contexts. This vision or philosophy is subsumed to a large extent in a concept introduced by the author in [Simos88]: an *organon*, a centralized repository of models, assets, generative tools and codified domain-specific knowledge that provides an evolving infrastructure for communities of collaborating technical workers. Organons will incorporate constructive, generative, knowledge- and model-based reuse technology, and will be structured by domain models that manage changes in underlying reuse technology so as to minimize disruption to the participant community. Organons will thus be reusable asset bases quite different in nature, though bearing some common aspects with, DBMS's, repositories of static components, catalogues, automated library systems, and generation systems. (The essentially hybrid nature of supporting technology for organons is analogous to the hybrid nature of the DA discipline itself, as suggested in Figure C.)

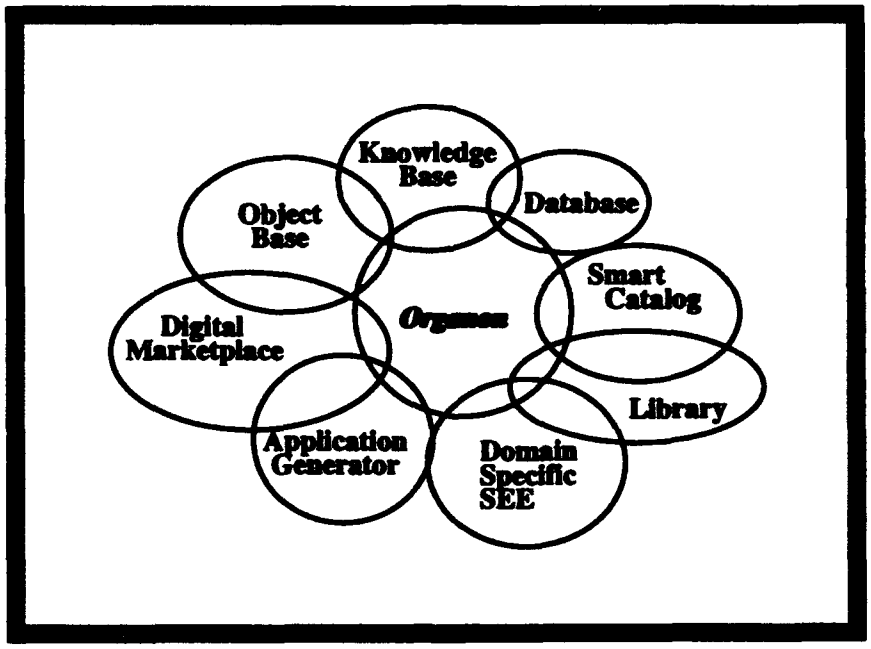


Figure C: Organon - Analogy Domains

Well-designed organons will be highly focused and special-purpose, organizing complex details about a domain within an intuitively graspable model. Hence most practical large-scale systems will be developed utilizing the resources of multiple organons. Careful scoping and boundary negotiation between organons for related areas will be a critical design factor. To effectively build and use organon resources, new technologies and better ways of integrating existing technologies will need to be developed; individuals will need to develop new kinds of software, process, and knowledge engineering skills; organizations will need to develop different group development disciplines, methods, policies and

structures; and the software industry as a whole will need to develop new market strategies and infrastructure. Many of these changes are already visible in the emerging vendor marketplace for software and/or information components of all kinds: extensible system platforms, tool kits, application environments, component libraries, object-oriented collections, and multimedia collections. In many cases, however, the development of the necessary technology has far outstripped our progress in the other enabling factors essential to making the organon vision practical.

ODM concepts and processes, while useful in the near term as a more formalized and methodical approach to DA, are intended to be robust and scalable enough to apply in the context of this broader reuse vision. While addressing a limited scope within the reuse life cycle, ODM attempts to treat the domain modeling process as an incubator context supporting balanced development of the necessary technological infrastructure, individual cognitive modeling skills, and organizational skills, management strategies and group forms and structures for reuse; to provide both a workable method of DA and a transition path to move our current approach to reuse technology towards the organon vision.

ODM's emphasis on careful domain definition and scoping is based on the expectation that domain resources will be accessed from a wide network of similar kinds of resources (as also reflected in the work that has been done on the ALOAF specification [STARS92a]). The ODM emphasis on collaborative development of domain models and ethnographic methods for gathering domain information reflect the view of an organon as a consensus-based knowledge management structure owned and utilized by a specific community of technical workers. ODM's emphasis on the combined use of inheritance-based, architectural and generative modeling techniques reflect the anticipated use of hybrid technologies within even a single organon, and the evolution of technology support within domains over their life cycle.

4.3 ODM and the CFRP

As part of the STARS-supported work on ODM, ODM processes and workproducts have been structured in accord with the reuse process model of the CFRP (Conceptual Framework for Reuse Processes). The CFRP has been developed as a key joint STARS product, with direct contributions from all three STARS Prime organizations (including participation from the author). The CFRP is defined in the context of a broader STARS vision of *megaprogramming* which integrates the three key themes of domain-specific, architecture-based software reuse, process-driven software engineering, and technology-supported development. The place of ODM in the context of this related work in reuse is depicted in Figure D.

ODM is strongly rooted in the process structures provided in the CFRP model. While the high-level organizational and technical vision underlying ODM (e.g., the Organon concept) is reflected to a limited extent in the STARS CFRP, neither the CFRP nor STARS in general necessarily endorse the Organon vision, with its ramifications for organizational design and change management. Linkage between ODM and the CFRP takes place on at least three distinct levels.

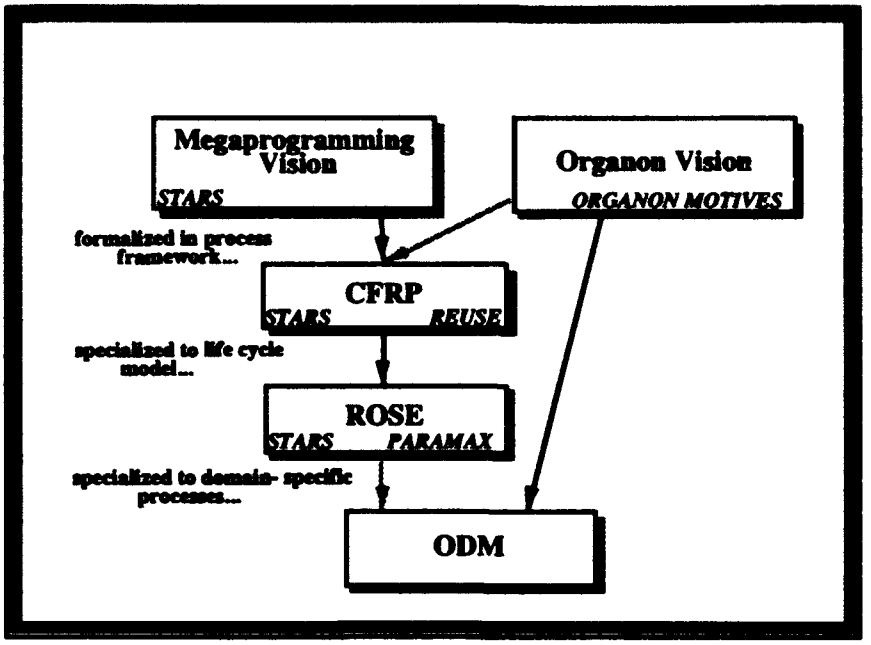


Figure D: ODM Context

First, ODM processes and workproducts have been mapped into the relevant process idioms, families and categories of the CFRP, as reflected in the structure of this technical report. ODM processes refine CFRP process categories concerned with modeling domain-specific information for reuse. In CFRP terms, the scope of ODM is primarily that of a single Asset Creation (i.e., domain engineering) project. ODM's domain planning and learning activities directly reflect CFRP process categories. The ODM method is based on a critical distinction between descriptive and prescriptive modeling. These modeling activities are orthogonal to the Domain Analysis and Modeling, and Software Architecture Development phases within the CFRP Asset Creation process family. These two complementary perspectives on the domain modeling process can be integrated to generate a number of specific planning approaches and process models for domain engineering projects. The mapping of ODM to CFRP process categories provides a basis for comparison on ODM with other DA processes, as well as a validation of the CFRP in its function as a kind of domain model for the domain of software reuse processes.

Second, detailed ODM process architecture has been structured in terms of three lower-level Plan - Enact - Learn loops, each of which is further structured into sub-processes. The CFRP's Reuse Management Cycle (Plan - Enact - Learn) is reflected at several lower levels of the ODM process model and in the structure of ODM workproducts. In at least one case (detailed domain modeling), these further structures also reflect Plan - Enact - Learn idiom specializations.

Third, CFRP idioms can be applied recursively to any core ODM process or workproduct, and are even reflected in ODM strategies for technology transfer, training, and ongoing evolution. In accord with this last level, an *ODM Handbook* could eventually be developed that includes a *starter model* for each model in the ODM workset. Each starter model might include domain-independent fragments to be incorporated into domain modelers' workproducts, tailored as needed to the individual project. At this stage of ODM maturity,

ODM technology developers supporting the ODM Handbook would be, in effect, fulfilling an Asset Management role for an base of reusable ODM assets; and will fulfill the Reuse Learning role by collecting project experience to evolve existing ODM assets and define new assets. The long term ODM technology transfer strategy thus depends on the formation of an *ODM Organon*, which makes supporting tools and assets available and serves as a model of the ODM development process.

Specialization and focusing of CFRP to domain-related activities (ODM). ODM specializes those categories of the CFRP process architecture where domain-specific modeling activities take place. Figure B represents the CFRP process categories covered by ODM: the first two categories within Asset Creation (Domain Analysis and Modeling, and Software Architecture Development), some initial planning stages of Application Generator and Component Development process categories, and those portions of Reuse Planning and Reuse Learning activities centered around Asset Creation. Thus, ODM does not cover all process aspects of the CFRP; in particular, Asset Management and Asset Utilization activities fall outside the process scope of ODM. In addition, ODM reflects only one possible specialization of those process categories it does address. Other DA techniques might interpret the processes involved in Domain Analysis and Modeling, and Domain Architecture Development, differently than does ODM.

CFRP Specialization to Software Life Cycle Models. As mentioned above, ODM can be viewed as a specialization of the CFRP for processes related to DA. As more complete life cycle models are defined using the CFRP as a framework, it may become clearer to model ODM as a sub-specialization of one of these more detailed (and more "committed") life cycle models. An example would be the ROSE model currently under development [???].

Project-specific tailoring of ODM framework to reflect specific DA plans. ODM is a "specialization" and not an "instantiation" of the CFRP, because it remains to some extent a domain analysis process "framework" rather than a specific set of steps in a defined order. Because ODM is intended to apply in a broad spectrum of organizational contexts, a number of the processes and workproducts called out by ODM might be performed and/or created in different sequences by different analysts in different contexts. (For example, if the domain selection is relatively unconstrained but the representative set of systems is highly constrained, the Domain Genealogy might be developed before Domain Selection.) As project experience grows, ODM materials (such as the "handbook" alluded to earlier) will evolve to include guidelines, lessons learned and heuristics about what process sequences to adopt in particular contexts.

Because of this inherent variability, tailoring of ODM into specific DA project plans and policies is not only possible but is an anticipated part of the process. For large organizations, tailoring might first take the form of a specific domain analysis life cycle process model that specifies certain process sequences left under-specified within ODM, discards particular workproducts considered inappropriate for the organization or the types of domains to be considered; or addition of other processes, models and workproducts to the DA workproduct set. Where there is no requirement for the establishment of standard procedures across several DA projects, analysts might "instantiate" ODM directly into the form of a DA Project Plan. Previous or current DA projects could be modeled for purposes of description, comparison and review in the same way. This tailoring and evolution strategy for ODM is depicted in Figure E.

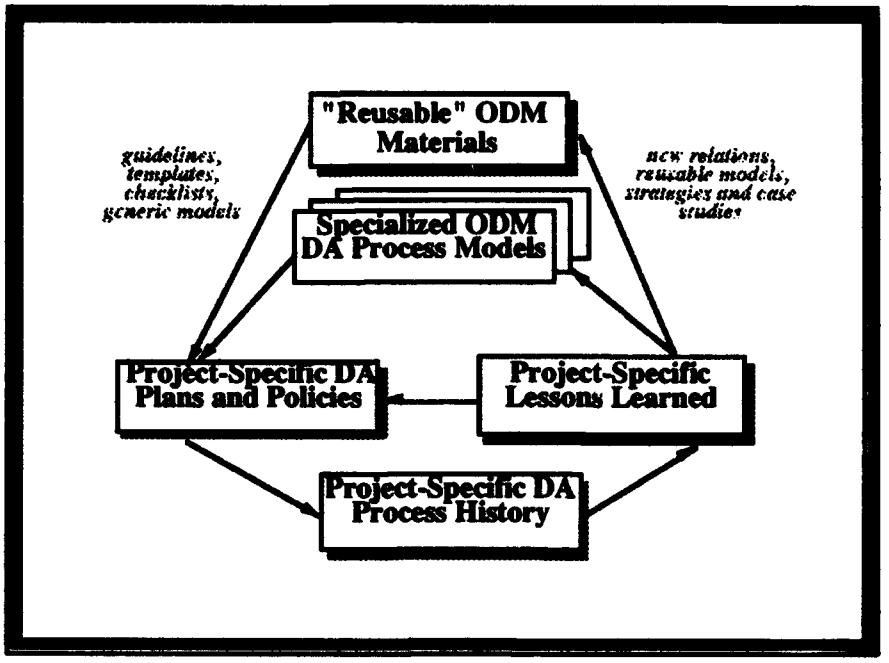


Figure E: ODM Tailoring and Instantiation

4.4 ODM and Inheritance-Based Modeling

As a specialization of the CFRP, ODM can be viewed as a as a series of domain analysis processes and workproducts. Particular modeling and representation formalisms are not integral to this process model layer of ODM. In fact, one process in the up-front Reuse Planning phase of ODM is selecting and tailoring the DA methods and modeling techniques most appropriate for the organizational context and characteristics of the domain. The ODM method recognizes a spectrum of techniques applicable to domain modeling, ranging from informal keyword-based classification schemes, to semi-formal approaches such as faceted classification, to modeling approaches incorporating inheritance (e.g., semantic network-based systems and object-oriented models).

However, the overall approach to domain modeling supported by ODM does particularly motivate an inheritance-based modeling approach. Other DA methods have emphasized simpler modeling techniques, such as faceted classification, or else advocate a wholesale adoption of object-oriented modeling techniques. ODM is oriented towards a form of modeling that could be termed *structured inheritance networks*. These provide specialization and aggregation (or part-subpart, composition relations) with a strict inheritance semantics defined on the networks produced. This strict interpretation of inheritance means that assumptions about parent concepts will always be maintained for their descendants. This attribute is of critical importance in domain modeling, where the models produced will eventually support assets in a structured asset base. The use of structured inheritance ensures that modelers and asset utilizers will be able to make strong assumptions about the behavior of assets, based on their placement within the hierarchy. Much of the complexity in ODM results in creating the necessary conditions for the methodical construction of these models.

While aspects of the ODM process model would be useful even for DA projects not employing inheritance-based modeling, the full value of the ODM approach can be best seen in the context of these techniques. Specialization and taxonomic relations are integral throughout ODM, not just in detailed domain modeling phases. Conversely, some aspects of the ODM method may prove essential to use inheritance-based techniques effectively for domain analysis and modeling (e.g., careful scoping and bounding of the domain based on specialization as well as component relations between domains). Specialization and inheritance allows modelers to build more complex models, enabling reuse in more complex domains; but a high degree of formality and process maturity is needed for successful modeling in such domains. Some evidence of this need can be found in the example of the numerous ill-structured class libraries now in existence, legacies of early and enthusiastic adoption of object-oriented programming languages without adequate accompanying strategies for developing, managing and controlling object libraries.

4.5 ODM and Specific Toolsets

The paragraphs above suggest ODM can be viewed as a layered model, as suggested in Figure F. The top layer of ODM is a core set of principles and concepts that underlie all the specific processes and workproducts. The primary focus of this report is at the second level, the ODM *process model* defined at a representation-independent level. A set of formal notations for various ODM workproducts in currently in development, provide a modeling framework that hides inheritance-based modeling at a lower, "detailed model design" level. A further, "ODM implementation" level can be specified in the context of specific tools, such as the RLF toolset. The compatibility of ODM and RLF-based modeling is not surprising, in part because ODM was first prototyped during the design of the RLF system, and in part because this system was designed specifically to support the kind of domain modeling emphasized in ODM. Thorough discussion of the potential for representing ODM workproducts and processes in RLF or other toolsets is beyond the scope of this report.

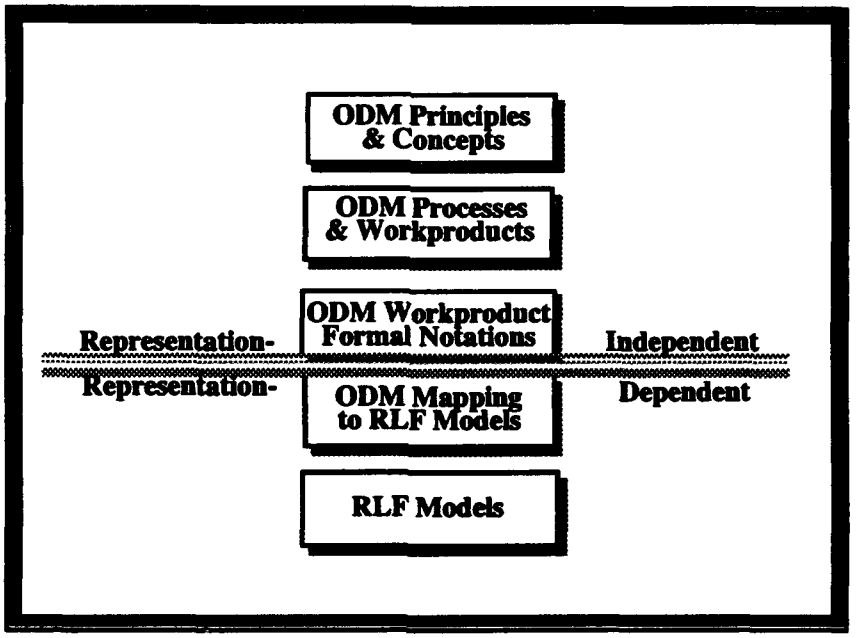


Figure F: ODM Layered Abstractions

5. ODM Process Overview

This section presents a brief overview of the entire ODM process model. Detailed descriptions of each process are provided in the main sections of this report.

When viewed as a whole, the ODM process has two major goals. The first goal is to structure the domain modeling life cycle as a series of incremental scoping steps; each step builds on scoping decisions made in previous steps and offers different opportunities for further refinement and focusing. This process is depicted in Figure G. The second goal is to provide a systematic transformation from artifacts to assets in a domain-specific context. These complementary views are described further in the following paragraphs.

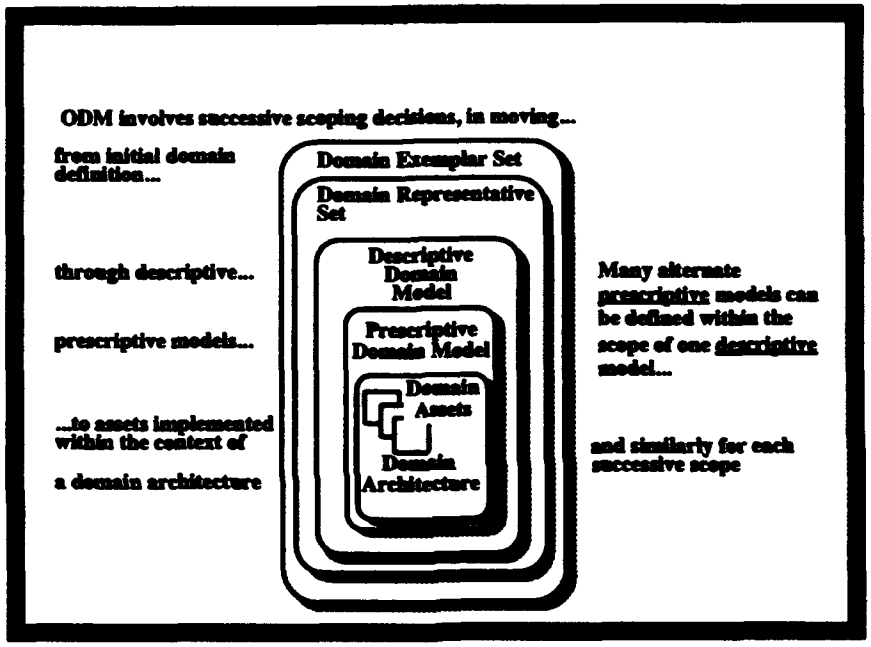


Figure G: ODM Process as Successive Scoping Steps

Successive Scoping. The initial definition of a domain in ODM is empirically grounded in a set of example systems, or *exemplars*, for the domain. A subset of these exemplar systems, the *representative set*, is selected for intensive analysis. These systems are analyzed, focusing on particular selections of artifacts from various system life cycle phases, resulting in a set of descriptive domain models. After the descriptive models are validated and refined, a new rescoping is performed to identify the range of functionality that will be supported by the reusable assets to be developed. This phase is *prescriptive* in that binding design decisions are made by the modelers. though these decisions will span multiple application contexts. The prescriptive model of functions (or *features*) to be supported leads to the development of a *domain architecture* to support the prescribed variants. After this architecture has been specified, asset implementors may develop only a subset of the assets specified in the architecture context. At every step of the process, scoping decisions serve as a risk mitigation strategy, constraining and managing the process. At the same time, the design and modeling decisions made at each point have a richer foundation of data on which to draw because of the previous steps.

Artifacts to Assets. Another way of viewing the overall ODM process is as a transformation or restructuring of artifacts from exemplar systems into *assets* in a managed and architecturally integrated asset base. The mapping takes place through the intermediary definition of *features* for the domain, significant differentiating capabilities across systems within the domain. Attempts to create asset bases directly out of artifacts lifted from single-application contexts of use have had limited success in previous reuse efforts. On the other hand, by preserving traceability from features back to exemplar artifacts, the ODM method provides asset implementors with possible prototype artifacts from exemplar systems on which to base the development of the new assets. Thus, the intent behind the ODM method is both to provide a systematic basis for reengineering assets for reuse, based on explicit documentation of their intended scope of applicability, and to maximize use of the existing legacy of system artifacts as a basis for domain knowledge and reengineering.

The following sections provide a fairly detailed overview of the ODM process. Each individual process and workproduct is described more thoroughly in Section 7.

5.1 Reuse Planning

ODM incorporates a subset of Reuse Planning activities specifically addressing areas of concern for Asset Creation; these are assumed to take place in coordination with broader planning activities for the entire reuse program scope. ODM planning activities include organizational assessment, definition of Asset Creation project objectives, domain selection, infrastructure planning and detailed project planning.

Organization assessment activities produce a general typology of the organization(s) within the reuse program scope; this typology is needed to appropriately tailor the ODM process. In addition, a Business Area Stakeholders Model is developed that structures potential asset creation, management, and utilization projects (within CFRP terms) within the program scope. This model and the more detailed Domain Stakeholders Model, is an important foundation for many later modeling processes, such as the Feature Binding Times Model (described in Section 7.2.3.4.1).

Domain identification, characterization and selection processes produce a Candidate Domains List, including characterization of several *domains of interest*, and a selected domain, or *domain of focus*.

5.2 Descriptive Modeling

The Descriptive Modeling phase of ODM commences the portion of activities that could be considered part of Asset Creation (in CFRP terms). This phase is itself structured into three major processes that correspond in a specialized sense to the Plan, Enact, and Learn families of Reuse Management. Similar processes can be identified for the Prescriptive Modeling and Asset Implementation phases (although ODM addresses only the planning activities for the last phase). This basic process structure, reflecting three recursive Plan - Enact - Learn loops, is depicted in Figure H. The steps within Descriptive Modeling are described below.

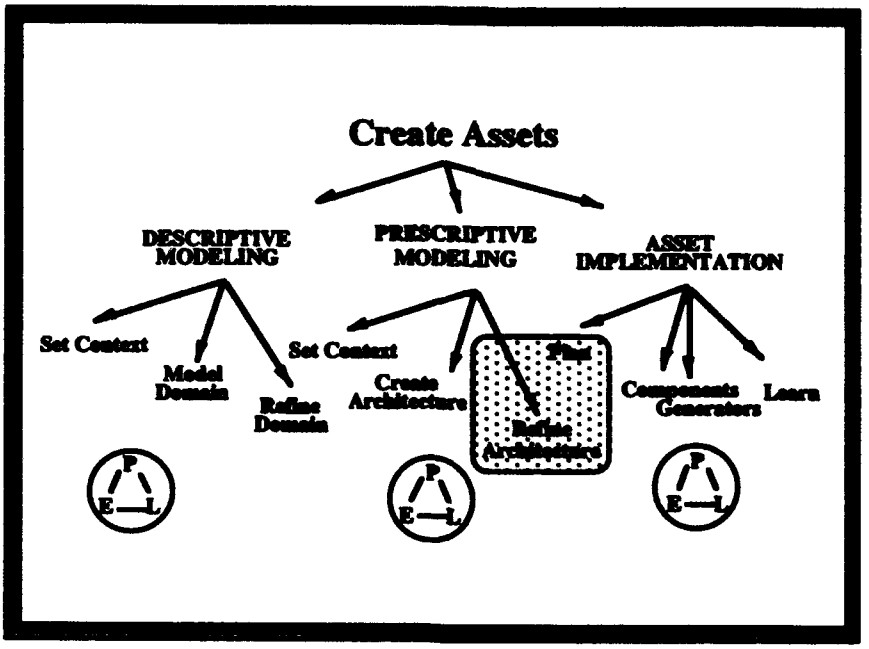


Figure H: ODM Plan-Enact-Learn Processes within Asset Creation

5.2.1 Context Setting

The first step in Descriptive Modeling is context-setting. The definition of the domain is formalized, both in terms of rules of inclusion and exclusion and in terms of identified member systems, or *exemplars*, from which a set of *representative exemplars* is selected for detailed analysis. A Domain Interconnection Model is created that documents qualitative relations between the domain of focus and structurally or conceptually related domains. Specific workproducts within representatives' software life cycle, models to be developed, and proposed representation methods and formalisms are specified.

5.2.2 Modeling the Domain

Descriptive model development actually involves creating a number of separate models to capture different aspects of the domain. Each model undergoes a similar life cycle of development, that includes the filtering of relevant domain information derived from documents or from interviews with domain informants. Representative exemplar systems (and/or subsystems) are investigated to identify a set of workproducts that will provide detailed sources of domain information. Traceability information is retained for these workproducts, which are in effect converted into *artifacts*, sources of domain information that may be used and/or interpreted in ways very different than their original purpose of creation.

Domain terminology is captured and managed in the Domain Lexicon, which documents the domain language as directly reflected in the artifacts and transcripts of informants' terminology. The lexicon helps to normalize domain terms at a syntactic or lexical level, keeping the descriptive models themselves focused on more essential semantic relationships. Lexicon terms are mapped into a model with more semantic relationships defined between terms. These models would ost appropriately be structured inheritance

network models, combining specialization or taxonomic information with aggregation or structural relations where these are integral to describing commonality and variability across the representative set. Separate models created in this phase are validated for internal and cross-consistency and for completeness and conciseness with respect to domain information, by applying principles of the modeling representation formalism selected for each model, by populating the models with representative data from domain artifacts, and by reviews with selected domain experts and other stakeholders.

The following are suggestions for useful ways of structuring further descriptive models into categories. Unlike many of the named models in this report, these categories are not required elements of an ODM approach. They are one way of approaching the transition from observable characteristics of exemplar artifacts to a model of distinguishing features for the domain.

Artifact Models: Descriptions of exemplar artifacts are formalized into one or more models of commonality and variability, the descriptive *artifact models* for the domain. Artifact models, somewhat analogous to the "current physical model" in Yourdon terms, describe commonality and variability in the domain exemplar information sources themselves (i.e., the artifacts). These artifacts are modeled to provide linkage back to the artifacts from other descriptive models, and to capture knowledge about the structure of domain artifacts that may be relevant for asset implementation. Not every information source examined needs to be instantiated in the artifact models, but they should encompass the diversity across the artifacts.

Conceptual Models: Artifact models are the source of a second layer of descriptive models, termed the conceptual *models* for the domain. Conceptual models describe commonality and variability in the various concepts expressed in domain exemplars. Conceptual models are still descriptive because they are confined to precedented terminology or functionality within the exemplar set, although this may include terminology for various conceptual entities (e.g., objects, operations, relations, constraints, and environmental characteristics) within the domain. Conceptual models may also address different conditions within the operational environment for the systems in the domain, where those conditions are significant for specifying the semantics of system behavior.

Contextual Models: A third class of descriptive models derived directly from the information gathering process can be considered contextual *models*. Contextual models may include models of development processes, rationale and decisions. They may be linked to either specific artifacts (e.g., a process history or rationale for a particular design document) or to elements of the conceptual models (e.g., rationale for a certain constraint on an operation).

One key integrating model needed in the descriptive phase is termed the Domain Stakeholders Model. This model identifies various contexts in which potential domain functions are performed; each context provides visibility and interest to various domain stakeholders. For example, whether a given function requires a coding change, a site-determined parameter, a start-up value for an interactive session or a dynamically specifiable default will have significant impact on the usefulness of that function to developers and/or users of the system.

5.2.3 Domain Model Refinement

During Domain Model Refinement, empirically generated descriptive models are integrated, validated and interpreted, again with respect to other models and domain informants. A primary integration method is the formal definition of domain *features* as descriptions of capabilities of interest within the domain, linked to the previous descriptive models. The resulting integrated Domain Features Model may then be transformed, using a variety of techniques during Domain Innovation Modeling.

Domain Model Integration

In the first refinement process, Model Integration, the separately developed artifact, conceptual and contextual models are compared and validated with respect to each other. Problems to be identified and corrected may include unintentional redundancies and repetition within the models, inconsistencies in modeling format or conventions, and gaps in coverage that were not included in any of the models but are deemed essential by modelers. This step can be compared to the software integration testing that follows software unit testing in conventional system development.

Descriptive Feature Models: As a key integration step, models of distinguishing domain features are derived from validated descriptive domain models (primarily the conceptual models). In general terms, a *domain feature* (hereafter feature) usually integrates different categories drawn from different conceptual models. As an example, a feature might be defined as an assertion about the behavior of the system under a given operation on a given configuration of elements (as defined in the object model), given certain operating conditions and certain constraints (as defined in other conceptual models).

Stakeholder Context: In addition to being linked to various conceptual models, each feature is defined within a given stakeholder context, which includes not only that of the system end-user but might include the developer or other people along the feature binding time spectrum for the domain, e.g., site configurers, field support personnel, etc. Thus a feature model can contain more than what are conventionally considered user-visible functional features. The exact semantics of what constitutes a feature will be defined for each domain modeling effort subject to the conceptual models developed and the various stakeholder contexts applicable in the domain.

Domain Model Interpretation—Feature Semantics

Initially each feature may be derived somewhat independently as a cross-relationship between other descriptive models. A further stage of model integration takes place when features are organized into semantically related models. Well-defined semantic relationships on features are defined in the conceptual models. These relationships can be used to validate (and possibly generate) semantic relationships defined between features themselves. To the extent that semantic relations between features are not wholly dictated by these linkages to other descriptive models, a set interpretation semantics can be used: that is, each feature is associated with the set of all possible implementations that would implement that feature. One feature can be said to *specialize* another when it can be satisfied by a subset of the satisfying set of implementations for its parent, where the converse will in general not be true. Each significant high-level feature typically forms the

root of a semi-independent domain-specific taxonomy of feature variants, based on a mapping of each variant to other aspects of the model.

Traceability is maintained from at least each leaf (i.e., most specific) variant to at least one exemplar artifact requiring or satisfying that feature variant as a capability. Each variant in the descriptive model must map to some exemplar instance of use (or instance of requirement), but the semantic relationships will usually span elements from multiple systems, and serve in a comparative capacity. Thus the Domain Representative Set provides a concrete, if indirect, basis for determining the scope of what appears in the feature model, since features are characterized in relation to descriptive artifact, conceptual, and contextual models that include only material traceable to exemplar artifacts. The output of this interpretation step for domain features is an integrated Domain Features Model. This model forms one component of the Interpreted Domain Model, which may include other interpretive additions to the initial descriptive model. For example, this model may include rationale for particular patterns of co-occurrence of clustering of features within certain artifacts and certain exemplars. The emphasis is still on understanding and interpreting existing feature profiles.

Domain Model Innovation

For all models prior to and including the Interpreted Domain Model, the focus of the modeling process is still descriptive. At this point in the overall ODM life cycle, the modeling process has shifted from domain information sources, to artifacts, to domain concepts, to individual features, to semantically defined feature relationships. Once features and their relationships can be worked with directly within models, various methods of transforming these feature models can be applied to elicit possibilities unanticipated by developers or users. This process, termed *innovative modeling,* serves as the transition from descriptive to prescriptive modeling. In the Domain Innovative Model, feature variants are not required to map to precedent-setting exemplars.

Examples of innovation techniques include: the orthogonalization of operations against objects; the exploratory shifting of a feature associated with one binding time across the entire binding time spectrum (as defined by the Feature Binding Times Model), cross-checking of objects and operations from analogy domains identified during Domain Scoping; or successive suppression of particular features from feature clusters that co-occur across many exemplars, to tease out possible useful functions in leaner versions of domain functionality. The Innovation Modeling process can also explore the architectural level for the domain, in terms of overall feature sets supported by different versions of the domain system/subsystem as a whole. The resulting Domain Innovation Model may contain feature variants and combinations that no longer exactly correspond to precedented feature sets within domain exemplars. This model also does not represent commitments on the part of the modelers to support all the features and feature combinations expressed. Determining the intended scope of feature coverage for the domain asset base is the task of Prescriptive Domain Modeling.

5.3 Prescriptive Domain Modeling

The descriptive modeling phase can be seen as a mapping from exemplar workproducts, to artifacts, to conceptual entities, to semantically characterized features. The innovative modeling phase acts as the bridge between descriptive and prescriptive modeling,

performing transformations on the feature model itself. In prescriptive modeling, we finally begin the process of directly specifying what functionality will be directly supported by assets in the anticipated asset base. Our focus accordingly shifts from domain informants to potential customers for our assets.

The Prescriptive Domain Modeling phase prioritizes the feature and architectural variants produced by domain description and refinement. Specifically, each variant is evaluated for usability, relative to a specified base of potential asset utilizer environments. In addition, features are evaluated for feasibility relative to the available asset implementation and asset management infrastructure. Features are re-clustered, re-organized and mapped to high-level asset specifications. These specifications may reflect significant restructuring from the system artifacts first studied in descriptive modeling. The process can likened to a "mirroring" function, wherein domain features are the transforming link from the domain artifacts to domain assets.

The prescriptive models can be specified simultaneously from a top-down and bottom-up approach. The top-down approach explores feature sets describing the domain system as a whole, whereas the bottom-up approach describes the variants to be supported for individual asset-level functionality. Commitments are made and iteratively modified to a particular range of intended customers an intended features to implement. At the end of the prescriptive phase, there should be traceability links demonstrating consistency across these two sets of scoping decisions, i.e., between anticipated asset utilizers and supported features.

5.3.1 Usability Analysis (Rescoping)

The Domain Innovation Model has produced some novel feature variants, intermixed with variants that map to exemplars (including known requirements as well as existing implementations). During usability analysis, the feature combinations in the innovative model are used for an exploratory search of potential customers for assets with the features described. The Domain Stakeholder Model is examined and expanded to form a model of potential "customer contexts" for reusable assets. Some contexts within the descriptive scope may be "de-committed", because of insufficient need for the functionality across the domain. With caution, other customer contexts might be introduced at this time; these new contexts may require more detailed characterization.

This activity involves a packaging tradeoff analogous to that in product family planning, ensuring that variants supporting particular combinations of features have sufficient demand across the intended range of customers to warrant the overhead of separate creation and configuration management.

5.3.2 Feasibility Analysis

The various feature combinations and architectural variants are also analyzed from the point of view of feasibility. This analysis considers the relative effort required to implement (still, if possible, avoiding pre-emptive commitment to particular implementation techniques) and to maintain separate variants within the asset base. Here, the "supply-side" cost of maintaining multiple variants is considered in a tradeoff analysis complementary to that in Usability Analysis.

5.3.3 Domain Architecture Modeling

The results of usability and feasibility analysis are synthesized into a model that represents binding commitments or specifications for functional profiles of domain assets. The output of this phase is an overall structure for the asset base, including a range of architectural variants to be supported by the asset infrastructure and a set of specifications for "asset ensembles". Asset ensembles represent functional areas within the domain that may be implemented with sets of static components, generative tools, or hybrid solutions; and which may be developed using a combination of newly developed workproducts and adaptations of existing artifacts characterized during the Descriptive Modeling phase. In ODM, the term *domain architecture* refers to this combination of an overall structure for the assets in the asset base, allocation of prescriptive features to asset ensemble specifications, and a high-level description of the abstract interfaces between the ensembles.

The architecture is built around these asset ensembles, rather than directly around more detailed asset specifications, so that the architecture is insulated to a degree from the effects of local changes in technology, such as conversion of an individual family of static components into a generative form. The domain architecture and asset base structure produced during the Prescriptive phase attempts to structure these ensemble groupings so that shifts in asset implementation technology will have localized impact on the asset base structure as a whole.

5.4 Asset Implementation

In Asset Implementation, alternative strategies for implementing the feature profiles allocated to each asset ensemble are explored and implementation strategies committed. Key issues to be determined in this exploration stage include tradeoffs between component-based, generator-based, or hybrid implementation strategies for particular asset ensemble specifications. In addition, possible leveraged development of the new assets can be considered, either from assets drawn from existing asset bases, or artifacts modeled during the descriptive modeling phase. The latter will typically require reengineering to conform to the asset specifications produced by prescriptive modeling In addition, tradeoffs in the structure of the asset base must be considered between strong coupling (with internal reuse) vs. loose coupling (with relative autonomy) of assets within the asset base.

Based on the strategy determined, an overall implementation plan for the asset base is produced, making optimum opportunistic use of generative capabilities, phased or evolutionary development, and bootstrapped use of early components later in the development cycle. A test and validation plan is produced which also takes advantage of the semantic relationships between asset variants within the asset base.

The ODM method does not address the actual construction of the assets (components and/or generators). By correctly following the steps described above, through the Planning phase of Asset Implementation, asset specifications should be produced that reduce the problem of implementing reusable assets to a tractable design problem. Here, the software development methodology used within the organization can be applied (with the proviso that this application engineering methodology should itself be adapted to utilize, where possible, existing workproducts).

Specific techniques required for engineering generator-based assets (or software in general) are outside the current scope of ODM. However, the ODM method is designed to create specifications that allow for consideration of both constructive and generative solutions, and later migration to alternative implementation strategies with minimal disruption to the overall structure of the asset base.

6. ODM Principles

This section presents a model of the ODM enabling disciplines, core concepts essential to an understanding of and competence in ODM processes.

6.1 Enabling Disciplines

Peter Senge, in **The Fifth Discipline** [Sen91], uses the terms *enabling technologies* and *enabling disciplines* to describe sets of independent but mutually reinforcing capabilities (including personal skills, organizational competencies, and supporting technology) that work synergistically to transform inventions into *innovations*: techniques or methods proven not only technical feasible, but economically sustainable, replicable and manageable. In these terms, it is fair to characterize current DA state of practice as in the "invention" rather than "innovation" stage. In particular, viewing DA as merely an extension to systems analysis has led to underestimation of the new set of skills and disciplines required to perform DA in a systematic and repeatable way.

ODM attempts to envision what a mature DA process will entail, and is founded on a core set of principles: essential skills, disciplines, practices, and perspectives for successful domain modeling. Skills utilized for DA differ considerably from those demanded of conventional software engineers, and that a sound plan for an ODM project will require some awareness of each of these principles.

As suggested by the name of the ODM method, these principles are concerned respectively with organizations, domains, and modeling. *Organizational* principles require awareness of the organizational context for the domain modeling effort, the interpersonal and group dynamic gathering and modeling domain information. *Domain-oriented* principles concern the nature of the domain modeling process itself. Many skillful software engineers and systems designers currently do domain modeling in an informal and intuitive way. While these methods can produce very useful results, the ODM method suggests modeling principles that can be applied to make this informal process more manageable, provide clear entry and exit criteria to various modeling stages, and differentiate what information belongs in what models. The third set of principles involve various *modeling* or model representation disciplines. While the ODM process model could be enacted without use of tools specifically supporting these representations, they reflect techniques for generating alternate views of domain information that are critical to the quality of the models. These core principles are depicted in Figure I, the central (unnamed) principle corresponds to the interaction and synergy of the other principles in creating innovative modeling transformations.

ODM can be thought of as a method on at least two levels: at one level, it provides a kind of "meta-process" for domain analysis that can include, as a specific step, selecting specific DA processes to apply. On another level, ODM represents a specific commitment to certain core modeling principles, such as the separation of descriptive and prescriptive modeling

activities. If the methods selected depart from these assumptions, there is of course no guarantee that the process model will be of use.

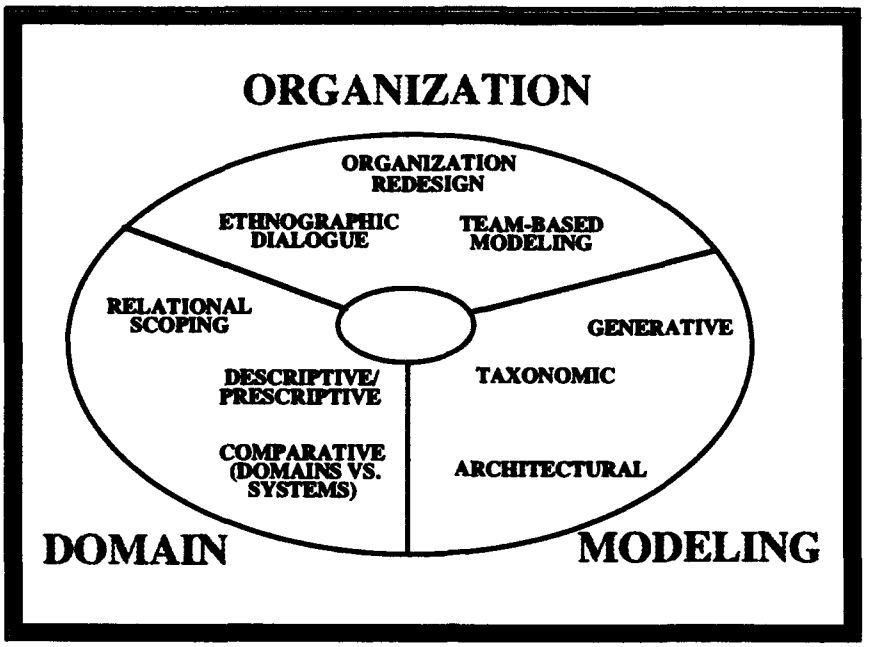


Figure I: ODM Core Principles/Enabling Disciplines and Technologies

All of the principles relate to *practices* in that: 1) they pervade all aspects of the ODM process model; and 2) they are evolvable, i.e., they will become more systematic and supported over time (as we practice them). The ODM enabling disciplines model can be used as a guideline for selecting good candidate personnel for DA efforts, keeping in mind that, at least initially, it will be rare to find individuals with strong skills across all disciplines. Therefore, ODM projects are best performed in the context of multi-disciplinary teams, working collaboratively on planning, information gathering and modeling activities. The enabling disciplines model can also serve as a basis for structuring training and team learning exercises, and as validation and quality control criteria for ODM workproducts. Each principle in the model is also a potential basis for improved automated support of the ODM method, and therefore represents either an existing or a potential *technology* as well. The following subsections describe ODM principles in more detail.

6.2 Organizational Principles

The organizational principles embodied in the ODM method are founded on recognition that DA is most effective when performed in the context of a broader transition to reuse-based software engineering, and that this transition requires accompanying intrinsic changes in organizational structure and culture. ODM therefore treats domain modeling as an indirect form of organization redesign. Because the shift to a reuse paradigm involves organizational change, general management theoretic approaches to organization design, change management, technology transfer, and organizational learning are relevant disciplines to reuse adoption.

At the same time, the reuse field, and specifically the discipline of domain analysis, has a contribution to make to general organizational development. The ODM method approaches the domain modeling process itself as an incubator for initiating organizational change. The qualifier "organization" in the method's name highlights the fundamental importance of understanding organizational context in domain modeling.

The principles described below—organization design, team-based modeling, and ethnographic dialogue—reflect implications of this general approach at the broad organizational level, the team level, and the level of individual modelers' interactions.

6.2.1 Organization Design

ODM provides new ways of studying and restructuring organizations in the light of specific domains of knowledge covered by its systems and personnel (i.e., its community of knowledge workers). In this sense, ODM can be considered an "organizational modeling" discipline–one that could eventually be integrated with broad-based strategies for organizational change and redesign. A key aspect of this discipline is the characterization of domains within their organizational context.

In the ODM approach, domains are viewed not as pre-determined classes of systems but as "designed abstractions". Domain models are not correct in and of themselves, but are instruments for certain classes of domain *stakeholders* ; within specific organizational contexts. These include, in the case of software reuse, system developers as well as end-users. Thus different modelers in different organizational contexts might define, scope and/or model the "same" domain (i.e., same functional area, and possibly even the same set of example systems, quite differently. Effectively, this would result in distinct organization domains.

Hence, whereas other methods refer to "problem domains", "application domains", "computer science domains" or "software domains", ODM uses the term organization *domain* to denote a coherent body of knowledge about a class of software systems or system functionality, defined and shared by communities of system developers and users within or across various stakeholder organizations. The "organization domain" notion strongly overlaps that of "core competencies" [Sen90, Pra90], a given enterprise's areas of strategic advantage, or "communities of practice" [IRL ???]. To provide strategic advantage, a domain must not only be an area of expertise for the organization, but this expertise must provide marketplace advantage in some way.

The diversity of stakeholders who, together, define a given organization domain (hereafter, the word domain will be used in its ODM interpretation) is integral to the domain's value to the organization. Domain knowledge is assumed to be distributed throughout the organization, not just within software engineering groups. For example, technical support personnel may contribute knowledge about typical ways in which end-users need to tailor the systems they receive, and this may in turn shape the defined boundaries of the domain model. Domain assessment and domain information-gathering from diverse sources of expertise within the organization are core aspects of the ODM method.

The modeling activity by which a community comes to self-identify around the domain concept can become itself an intervention in the dynamics of the organization. The more diverse the group, the more conventional information disconnects and rigidified producer-

consumer relationships can be realigned through the modeling process. Where the domain is of strategic importance to the organization, the domain modeling process can serve as an impetus to larger-scale restructuring and integrating processes within the organization.

6.2.2 Ethnographic Dialogue

ODM assumes active collaboration among domain experts, application engineers and internal or external reuse consultants in developing domain models. Through the reflective process of modeling formerly tacit, undocumented knowledge, and the experience of reconciling the knowledge of multiple experts, the domain modeling process itself can have a transforming effect on domain experts' knowledge of their domain and access to that knowledge, hence on the organization as a whole.

For domain modeling to have this effect, domain modelers must be able to practice a set of skills and techniques that can be termed *ethnographic dialogue*. Ethnographic techniques are employed by social scientists to discover qualititative and/or informal and undocumented knowledge about a given social situation. These techniques have been applied by researchers in technology-intensive workplace settings [Suchman?, IRL?] They are particularly valuable in a domain modeling context because much domain knowledge exists in undocumented and unsupported form, literally as "techlore" within the shared war stories and techniques of veteran engineers.

An academic ethnographic approach, however, will probably not be successful in transitioning the results of domain modeling into the organizational setting of interest. For successful technology transfer, the modeling activity must be perceived as participatory and dialogue-based for both the application engineers and the modelers and/or reuse experts interacting with them. The descriptive modeling principle (discussed below) allows domain modelers to approach interviews with domain experts and other informants with a relatively non-judgmental attitude. The modeling process is shared, at least in regular phases of review and co-creation, with those knowledgeable in the domain. The long-term goal is to enable application engineers to do their own modeling under their own initiative. This paradigm of dialogue-based modeling has important implications for the potential role of domain analysts or reuse consultants as interveners or change agents within organizations, working collaboratively with domain experts and application engineers in developing and evolving domain models.

6.2.3 Team-based Modeling

Many aspects of the ODM process require different sorts of team interaction skills than those customary for groups of software developers. For example, the modeling process works by continual interaction between centralized and distributed modeling tasks. Several groups may build related models relatively independently, then meet together to integrate the models and resolve the inconsistencies. Unlike software modules, that can have an unambiguous interface that creates a clear separation of concerns at the outset, related models will tend to overlap and interpenetrate. Especially given the current stage of maturity of ODM, where the process models inevitably leave much to be individually negotiated, this requires a high degree of constant, detailed technical interaction on the part of the modeling teams. This is a significant new kind of discipline to develop.

6.3 Domain-Oriented Principles

The three domain-oriented principles of ODM described below each involve a key conceptual skill in modeling domains. Each of the three principles is concerned with the ability to maintain a separation of concerns. The first involves distinguishing between domain modeling and system modeling content. The second principle relies on a further distinction made between different phases of the domain modeling process itself. This distinction can be characterized roughly as a separation between *descriptive* and *prescriptive* modeling. While this distinction is well-known in certain social scientific areas, it appears to be, at present, a unique aspect of the ODM method as applied to domain modeling in a software engineering context. The third principle involves distinguishing boundaries between domains, and characterizing the inter-domain relationships in ways that lead to well-modularized and coherent domain models.

6.3.1 Domain vs. System Modeling

ODM establishes fairly strict criteria and guidelines for what content should be included in domain vs. system models. Because of this rigorous separation of domain and system modeling, the ODM approach can be integrated with a variety of software analysis and design methodologies. Learning to discern this separation of concerns in practice calls upon a new set of conceptual skills for the prospective domain modeler. Software engineers design by instinct, and therefore the easiest mistake to make in domain modeling is to intentionally or unintentionally propose a specific system model as a domain model.

In ODM, domain models focus on describing commonality and variability across features of different systems. Functionality common to all systems in the domain can be incorporated in the intensional definition. Unless a semantics of inheritance can be clearly defined for the various components of a given model, the model is included in ODM as a specific component or asset, rather than as part of the domain model set itself.

For example, consider a data-flow diagram of a portion of a system in the domain. Many domain analysis methods advocate inclusion of such a workproduct (perhaps at a high level of abstraction) within the domain model. However, the artifact in question was developed within a single-system context. It therefore does not include elements that are intended to describe *comparative* information derived from the study of **multiple systems**. If the workproduct is abstracted to the point that variability across systems becomes invisible, then presumably it could be used as an asset, without modification, for any system in the domain. Assuming this is true (a point open to question) ODM takes the position that the workproduct could be included as part of the domain definition itself (i.e., as a rule of inclusion for the domain) or encapsulated as a potential asset.

Aggregation and decomposition relations are necessary in descriptive domain modeling, but are used to identify potential points of commonality and variability at all structural levels of the system. Aggregation will be particularly important when considering variant architectures and separately selectable subsystems for the domain, in the Prescriptive Modeling phase. But the integrity of the method requires clear separation between these modeling tasks as opposed to models whose intent is to describe the structure of a single system. ODM in fact assumes that system modeling methods and workproducts of this kind will be available for the domain, and views the domain modeler's task as describing, in one sense, relations *between* these system models.

System modeling can even be done within the scope of the domain modeling project, if necessary, to *normalize* artifacts from different systems into analogous form. This activity would still be considered system modeling. Systems analysts study the information flows and processes in the user environment, i.e., the operational behavior of the system being built. The end-user environment is usually conceived in isolation from the developer's environment. The connection between these two environments show up in terms of technology transition plans, deployment strategies, and technical support and maintenance relationships.

Domain analysis presents problems for this traditional systems analysis approach in several ways. First, in domain analysis, multiple end-user contexts must be studied in parallel. Scoping the set of end-user contexts to study is itself a dynamic and difficult process. Second, if design for reuse is to consider more than just static components, then the processes performed by the developer must be studied, almost in a similar manner to the way the end-user environment is studied in conventional software engineering. Finally, even the notion of two distinct environments (developer and end-user) turns out in most cases to be a dramatic oversimplification of the interactions between various producers, value-added re-sellers and consumers of software "layered" products. Without an integrated approach to studying these environments, we risk missing two important sources of innovative ideas for reuse engineering: the processes of the developer; and the potential reusable workproducts manipulated by the end-users themselves.

ODM shifts this perspective in several respects: first, by viewing software development artifacts not solely from the point of view of structural or architectural decomposition (e.g., module call graphs), but from a taxonomic or classificatory perspective; second, by focusing on the artifacts and processes of the system development environment. Thus, where a systems analyst would model a hospital environment with entities corresponding to invoices, payments, and medical reports, a domain analyst might also examine flows of information involved in **building** hospital software systems: requirements documents, designs, code, test plans, site-specific constraints, etc. (However, these separate views would appear in dinstinct descriptive models, partitioned according to the various *stakeholder contects* in the domain.

6.3.2 Descriptive vs. Prescriptive Models

Definition

The ODM method encourages a clear distinction between what are termed "descriptive" and "prescriptive" models. This distinction is easily confused, but independent of, distinctions such as between the problem and solution space for a domain. Most engineering design processes we are familiar with are inherently prescriptive: they describe how systems could be arranged in a hypothetical world of design alternatives.

The descriptive stance is something like that of a "software archeologist" or "techlorist"-- noting with interest what existing systems do without rushing too quickly to propose our own new designs. A descriptive model is empirically based on the common and variant features observed in an explicit set of domain *exemplars* studied as part of domain modeling.

This discipline can help offset the risk of "creeping feature-itis" (a phrase due to Bill Frakes), a phenomenon where in the design of a supposedly generic or universal system, or in the face of intense competition among products characterized in the marketplace by options and features, we begin to indiscriminately add features without regard to the usability of the overall feature set or the price of creating such superset functionality systems.

Rationale

The distinction between descriptive and prescriptive modeling is one strategy to avoid the recurrent problem of domain analysts imposing hidden system design decisions under the guise of domain analysis. Even domain analysts who are not domain experts are often sorely tempted to leap to what is obviously the best (i.e., most reusable) generic architecture or set of assets for the domain. In conventional systems analysis, we can validate such intuitive design decisions by mapping them to a specified set of requirements for the specific system of focus (although this procedure can guarantee only an adequate, not a "best" solution). In domain analysis, released from the constraint of meeting a single system's requirements, we are in danger of making these decisions in a kind of vacuum.

Compounding the problem is the fact that historically reuse initiatives have been coupled with other technology innovations, such as the introduction of Ada and/or object-oriented design methods. These methods often have strong prescriptive content around what kinds of architectural solutions to apply. Hence, not surprisingly, many so-called "domain architectures", in this author's opinion, are really object-oriented architectures for systems in the domain, derived without any systematic linkage to an existing set of systems. The point is not to dispute the relative advantages or disadvantages of such design methodologies, but rather to "unbundle" them from the domain analysis process itself.

Uses of the descriptive model

By studying the artifacts and architectures of historical and existing systems, we learn what issues need to be faced in reengineering these legacy systems to accommodate the reusable architectures and assets we develop. The descriptive modeling of legacy systems can in fact be viewed as a form of reverse engineering or system understanding for these systems. The descriptive models can therefore be of direct use to reengineering efforts for these systems. Furthermore, planning a domain analysis project in coordination with such a reengineering project may provide many valuable opportunities for synergy in joint tasks and shared workproducts. (See the relevant CECOM Demonstration Project report. [???])

Since every feature variant appears in at least one version of an exemplar system, feature combinations generated from this descriptive set of features have, in effect, been partially prototyped by existing systems. This contributes to the overall feasibility of the proposed assets to be implemented, and can even provide links enabling asset implementors to examine existing artifacts as a basis for their implementations.

Since development of reusable assets will itself be accomplished through a combination of new development and reengineering of existing exemplar workproducts, the descriptive phase can also serve as an inventory of the resources available to the asset implementor from existing systems. This does not imply that workproducts from legacy systems can be reused in a managed and systematic way without careful reengineering: there seems to be consensus in the reuse research community that assets must be designed for reuse rather

than simply pulled as is from single-system contexts. But such assets also need not be built entirely from scratch; doing so would be an ironic failure of reuse technology to apply its own philosophy to its own work processes.

Distinguishing Descriptive-Prescriptive

The distinction between "descriptive" and "prescriptive" models is orthogonal to the more common distinction made between a focus on the problem space vs. solution space models. It is easy to associate descriptive modeling with the modeling of system requirements (what the system is supposed to do), and the prescriptive modeling with the modeling of system solutions and designs (how the system will do it). However, we can build both descriptive and prescriptive domain requirements models.

A descriptive requirements model would describe all requirements seen in the workproducts of a specific set of systems in the domain (*exemplar systems*). The prescriptive requirements model would choose (i.e., "design") the specific set of requirements to be included as assets in the asset base, or that will be satisfied by assets in the asset base. Similarly, one could build both descriptive and prescriptive architecture models for a domain; the descriptive architecture model would describe the range of architectures existing in the domain, while the prescriptive model would determine the range of architecture variants to be accommodated by the asset base.

There are many other potential points of confusion around what descriptive vs. prescriptive modeling is, and how it compares to other approaches to partitioning the modeling process for domain analysis. A full discussion is beyond the scope of this report.

6.3.3 Relational Scoping

Software design is always done for a specific context or range of applications, whether explicitly or implicitly defined. In design for reuse, this context is usually broader than that of a single system or application, but it must still be defined; one cannot design for reuse by trying to specify an "optimally generic" component.

A fundamental principle of ODM, and one that requires a surprising degree of care in practice, is to explicitly define the intended scope of applicability for each model and asset developed in domain engineering. This bounding process is done in a number of ways: through definition principles of inclusion and exclusion, designation of a set of exemplar systems, and, in a later refinement step, characterizing the relationship of the domain of focus to other domains with which it shares interfaces or cognitive connections.

The latter concept, that of *relational scoping*, depends on an understanding of the nature of domains that violates our intuitive analogy to "objects". Objects are usually imagined as distinct and non-overlapping entities, existing in some space that separates one object from another without itself being an object. This is not an appropriate analogy for domains, which represent arbitrary (i.e., designed) partitions of a continuum of existing systems, artifacts, and individual stakeholders according to some defined principle. Thus, in effect, there are no "spaces" between domains; every boundary of a domain is an interface to what could at least potentially be defined as a related domain. Domains can intersect, overlap and interpenetrate, since they represent principles of membership. (Of course, one goal of careful domain scoping is to make sure our domains do not unintentionally overlap in awkward ways.)

Bias Towards Well-Scoped and Modular Domains

ODM encourages a conceptual bias toward definition of well-scoped, cohesive and highly scoped domains that can appear smaller relative to what are called domains in other methods.

Domain models developed in ODM are semantically rich and complex. The method works from the assumption that a given domain will involve much more detailed knowledge than suspected by any other than application specialists (who may themselves underestimate the degree of knowledge they possess—until they try to formalize and describe this knowledge to an outsider). The complexity arises from a focus on multiple systems as exemplars, the need to capture underlying design process and rationale for artifacts, and the potential use of any workproducts from across the software life cycle as assets.

The implication, from a project management point of view, is that if a domain engineering project based on a given number of personnel and a given time span is compared to a conventional application engineering project using comparable resources, the scope of the domain selected should be significantly smaller than that of the system addressed by the analogous team. In particular, if the domain engineering and application engineering project are planned in coordination, (with roughly equivalent team sizes and resource constraints) the domain scope might reasonably address only a subset of the scope of the entire system.

Another reason to stress small, highly cohesive domain models concerns the anticipation that systems will be composed out of numerous domain-oriented asset bases. If each domain model tries to wrap an entire application area of concern into its scope, opportunities for cross-family reuse will be less tractable. We will thus lose much of the potential marketplace for the investment in reuse.

One paradox of reuse is the fact that a reusable design of sufficiently high quality may yield assets usable in a number of contexts beyond those specifically planned for. When we carefully declare our intended context, we do not only make reuse within that context more repeatable and predictable; we also enhance the ability of reusers to make use of the assets outside their intended context, because the contextual assumptions are captured and can be viewed strategically. Ironically, it is by trying to make something "reusable" in the abstract that we risk implementing an asset that will be "equally un-reusable" in a variety of contexts.

6.4 Modeling Principles

Modeling is a core discipline and a core technology in ODM. In this respect, ODM could be considered an adaptation to domain engineering of a general approach of model-based development [???]

In an ODM project team, *everyone on the team models.* Rather than viewing modeling as technical development, planning and process reflection as management, both aspects of the project involve modeling activity. This is related to the closely intertwined nature of reuse engineering and management activities.

Furthermore, *modeling starts at the beginning of the project.* Organization profiling, domain assessment, and context-setting all potentially involve the development of semi-formal models. This approach provides a natural training context for use of modeling formalisms and tools, as well as a practice setting for the interactive skills required in team-based modeling.

Three specific modeling techniques called out here have special significance in ODM. Each technique—taxonomic, generative, and architectural modeling—can be associated with particular formal representations or tools, but has more general relevance to the method. They can be thought of as conceptual modeling skills that are supported more or less directly by the technical infrastructure available.

6.4.1 Taxonomic Modeling

Domain models represent *relationships* between aspects of systems rather than the structure of a single system, or development *processes* by which system artifacts are created. A common requirement for all domain models is therefore a representation that allows differences to be explicitly notated.

While ODM is oriented towards the use of inheritance-based modeling techniques (such as semantic network-based knowledge representation) in building domain models, experience suggests that modeling can begin much less formally, with simple tools (e.g., a good text outline editor for representing both taxonomic and aggregate relations) and an emphasis on insights created through the modeling process itself. However, taxonomic modeling, whether fully or shallowly supported by the modeling tools, is a core aspect of the ODM conceptual approach. This kind of modeling can be unfamiliar even to experienced applications developers, because taxonomic models are rarely used in any direct way in developing single systems. (Note: this is not because taxonomic relations have no relevance in a single-system context, e.g., customer types, account types, etc. Newer methods like semantic data modeling, knowledge-based systems design and object-oriented technology do make extensive use of taxonomic and/or inheritance-based techniques. However, for many of the more mature domains in which DA will be applied, these concepts will be new to the application development groups.)

The fundamental importance of taxonomic modeling is revealed in the use of modeling techniques even outside the descriptive and prescriptive modeling steps proper. For example, the typological profile performed during organization assessment and the Domain Interconnection Model created during Domain Scoping each have embedded relations that are essentially taxonomic.

6.4.2 Architectural Modeling

Every specific system architecture must choose to centralize certain functions into separable subsystem-level components, while other functions are distributed throughout the code. For example, in one system security processing might be an independent function, invoked as a set of services, while in another system in the same domain security functions are scattered throughout the code. There is no intrinsically "right" or "wrong" principle to apply to evaluating these design choices. One objective of ODM is to reverse engineer such choices into a representation where even distributed functions within existing architectures are investigated for possible reuse engineering.

An intrinsic part of this reverse engineering is to attempt to recapture the rationale that led to the original architecture and design decisions in the example systems. If the DA team too readily adopts the accepted, traditional architectures within the business area as the structural basis for their domain partitioning, then the opportunities for re-engineering based on these hidden horizontal domains may be lost.

6.4.3 Generative Modeling

Both constructive (i.e., component-based) and generative software engineering techniques are as old as the field of programming itself. In light of this fact, it is remarkable that few if any software engineering methodologies have presented an integrated method for applying both techniques systematically and opportunistically to a given software design problem. Use of generative solutions on practical projects is widespread, but often happens despite rather than because of the methods in use. The reuse field has reflected this state of practice, in that different researchers adopt methods biased towards one style or another of encapsulating reusable functionality.

The ODM method is intended to foster systematic consideration of a spectrum of asset implementation techniques not available to the conventional applications developer. In particular, the reuse engineer has the opportunity to build solutions for both the developers' and users' environments. This makes an ability to transform given solutions alternately into component-oriented or generative form a core skill for domain modelers. Most domains will offer opportunities for both techniques. It is important to be alert for generative opporutnities at all levels of the system, from the component to the architectural level and across all life-cycle workproducts. For example, generative techniques are often useful in test case generation.

Constructive and generative approaches to reuse differ in more than technology. A fundamental cognitive shift is required to see opportunities for generative reuse as an integral part of software engineering practice. Software engineering processes as well as products must be studied, since generative solutions often involve codification of component tailoring and adaptation processes in the form of an automated tool (i,e., what questions must be answered in order to specify a generated component instance).

This kind of analysis cannot be performed on workproducts alone; ideally it should involve both intensive interviewing and direct observation of software engineering practice. Researchers have applied social scientific and ethnographic techniques to glean this kind of information from studying the workplace. ODM adapts these methods to the aims of domain analysis, studying the workplace of software developers as well as system end-users as an integrated system (using ethnographic techniques as described above).

6.5 Innovation Modeling: The "Tenth Pillar" of ODM

The three sets of principles enumerated so far—organizational, domain definition and scoping, and modeling principles—are distinct yet closely interwoven in practice. For example, the interactional principle of ethnographic dialogue is essential to the perspective from which descriptive modeling can be performed, while taxonomic modeling techniques provide a comparative language for such descriptive analysis.

A project team competent in all these principles could perform domain modeling using the ODM approach. Resulting architectures, designs and reusable workproducts, hopefully,

would have a strong likelihood of being utilized extensively by the intended base of customers (i.e., application developers). This level of eventual utilization, of both models and assets, is the ultimate success criterion for any reuse method or technique.

There is, however, one missing ingredient from the set of enabling disciplines described so far: the potential for revolutionary, as opposed to merely incremental, design innovations within the domain. Talented and/or inspired system designers continually make ingenious discoveries and innovations during technical development. Software design methods rarely address this part of the process, serving primarily as a way of rationally integrating and validating design ideas that may have been arrived at in a very intuitive way.

The problem in applying unconstrained, intuitive invention in the process of domain analysis is that domain analysts inadvertently impose hidden assumptions and context-specific solutions on resources that must serve a much broader and more diverse community of stakeholders than those of a single application. ODM attempts to address this problem by carefully distinguishing descriptive from prescriptive modeling processes. However, the intent of the method is to foster, not discourage, innovation. In fact, many of the processes and workproducts of ODM are structured specifically to provide a framework for a structured and manageable form of design exploration that can be termed *innovation modeling*.

The core of ODM's approach to innovation modeling techniques consist of a repertoire of *model transformation* techniques. Between the descriptive and prescriptive phases, a transition step is allowed for (using resources allocated at the discretion of the ODM project) that applies selected transformations to the descriptively based models. In the start of the prescriptive phase, the resulting innovations are evaluated for usability and feasibility.The transformation techniques rely on key abstractions in the ODM method: the notion of features, the use of analogy domains, the definition of various stakeholder contexts for the systems in the domain, etc. These concepts are explained more fully in the main sections of this report.

Innovation modeling does rely on the synergy of the other ODM principles described above. Organization redesign brings diverse stakeholders within the organization together as a community of interest around a specific domain of knowledge. Ethnographic dialogue techniques foster the exploration of terminology clashes and multiple perspectives among both domain experts and domain modelers. Team modeling processes maximize the creative interplay of these perspectives in producing domain models. Careful domain definition and scoping activities serve as focusing and constraining mechanisms, within which innovative techniques can be applied without the risk of uncontrolled explosion of complexity. Taxonomic, architectural and generative modeling representations provide a basis for modelers to practice seeing alternative ways of modeling the same information. This systematic exploration of alternatives should produce better-quality models and reusable assets.

Innovation modeling is not, strictly speaking, a "core" principle, in that a viable ODM process can be performed without this step. However, the hypothesis behind the ODM method is that innovation modeling will come to be perceived as intrinsic to the value of the domain analysis process.

7. ODM Processes and Workproducts

The following sections provide detailed descriptions of the ODM processes and the workproducts created as part of these processes.

The ODM Process Tree. ODM processes are presented in a hierarchical tree structure. The semantics of this tree can be viewed in several ways:

1) as a conventional process decomposition that could be mapped into a process model such as SADT;

2) as a taxonomic classification of domain engineering processes that could be configured in a variety of sequences; or, as a mixture of these two interpretations;

3) as a set of choice points where project managers will typically have options to select either sequential or parallel enactments of the processes involved.

In some cases, pre- and post-condition process dependencies will constrain the process sequences that can be instantiated. But there is a surprising degree of flexibility in the process structure, due in part to the impact of variant organizational contexts and assumptions.

Sequencing of modeling activities in ODM is often intended to support certain intuitive processes on the part of the modeler. Workproducts serve as ways of check pointing processes to allow for the capture of process history and decision rationale, and recommendations for improved practice as an integral part of each step in the method.

Recursive and Reusable Processes. Some processes in ODM are inherently recursive. For example, the domain selection task in Reuse Planning could yield a broad domain scope, which is further refined by a lower-level domain selection task. At any level, the "business area" is considered to be the enclosing application area scope from which the domain of focus will be selected; in principle, a domain at one level of planning could become the business area context for a lower level selection process. However, the domain scoping activity within Context Setting is far more than just a recursive repetition of Domain Selection; the activities are qualitatively different. In general, processes explicitly mentioned in the process tree are not merely *recursive variants* of other processes in the tree.

Readers will also find process descriptions that are reusable, or are reused within the scope of ODM itself. For example, selection processes occur throughout the process model, with a recurring set of sub-processes: reusing the available starter model, defining criteria for candidates, generating a candidate set, establishing characterization criteria, characterizing, ranking and prioritizing the candidates, documenting the selection rationale. Model validation processes also occur throughout the process. The approach taken here is to centralize these process descriptions wherever possible, even when this obscures the direct sequential flow of the process descriptions. For example, general validation processes are described under Reuse Learning, though they could take place after any modeling step.

For each step, the subsections below will describe the purpose and primary objectives of the step, inputs from other steps, models and other workproducts generated, aspects of the process itself, and guidelines and issues to consider (where appropriate). Models and other workproducts are allocated to the processes that appear to be the main locus of their

development effort. Often earlier versions of the models and/or workproducts will be initiated much earlier in the overall ODM process.

For many of the models and workproducts, there will be "starter" versions available as part of the infrastructure support for the method, particularly as the toolset matures. These are sometimes referred to as *starter models*. Some of these existing or potential ODM resources are mentioned in relevant sections, and some examples are included in the lexicon appendix to this document (Section D.5) *Resources* represent documents, tools or guidelines that would assist a developer in performing a given activity, but are not essential to its performance. This differentiates resources from inputs to processes. Resources would typically, but need not necessarily be, supported assets within some asset base containing elements useful for the reuse infrastructure.

There are a number of contexts within ODM where it is anticipated that different organizations might perform steps and processes in distinct ways. Where such alternative options can be linked to identifiable characteristics of organizations or scenarios within which DA takes place, ODM offers a strategy model that details variations in process sequence that might be appropriate for different situations. As an example, an organization which has not yet determined a domain, yet has a fixed set of example systems within its scope of responsibility, may decide to do the Domain Exemplar Set selection before Domain Selection proper, and in fact to let the selection of the exemplar set guide and constrain the domain selection process.

Assumptions. The following process descriptions assume that the domain engineering project (hereafter ODM project) takes place as part of a larger *reuse program* within an organizational context focused on a general *business area* or line of business. A number of potential *domains of interest* may be identified within this business area, out of which a subset will be selected as the *domain of focus*. Subsequent domain modeling projects might be reinitiated for the same domain, a sub-domain, or another domain within the same business area.

Each ODM *project cycle* consists of several *phases*: Reuse Planning (from an ODM perspective), Descriptive Modeling, Prescriptive Modeling, Asset Implementation, and Reuse Learning (also from an ODM perspective). As noted elsewhere, the ODM process tree does not cover all aspects of Reuse Planning or Reuse Learning, nor does it encompass the latter part of Asset Implementation. The five phases can be mapped to an overall Plan - Enact - Learn cycle (or Reuse Management cycle, in CFRP terms).

The three central ODM phases, Descriptive and Prescriptive Modeling, and Asset Implementation, are each further decomposed into Plan, Enact, and Learn *sub-phases* that are characterized more particularly within each process description. Each sub-phase, in turn, is decomposed into several (usually three) ODM *processes*; each process, in turn, may consist of a number of potential *tasks* or *activities*. (Note: this terminology may not be applied uniformly within the process descriptions below.) Only the decomposition down to the level of ODM processes is considered the formal part of the ODM Process Tree.

7.1 Reuse Planning

The Reuse Planning phase of the ODM Process Model is a direct specialization of the Reuse Planning categories of the CFRP Reuse Management idiom. Reuse Planning involves planning for all three Reuse Engineering process families: Asset Creation, Management and Utilization. ODM focuses on planning tasks directly related to domain-specific issues, but requires other domain-independent but organization-specific information as a strategic basis for domain selection and other domain modeling decisions. Most DA methods include domain selection at some degree of detail within the process model. In practical settings domain selection often involves business criteria and decision-making at higher organizational levels than that of the domain analysts.

Planning activities discussed here form a thread through a wider set of overall planning processes. The restriction of scope intended within the ODM context for each CFRP process category, illustrated in Figure J, will be briefly noted for each section below.

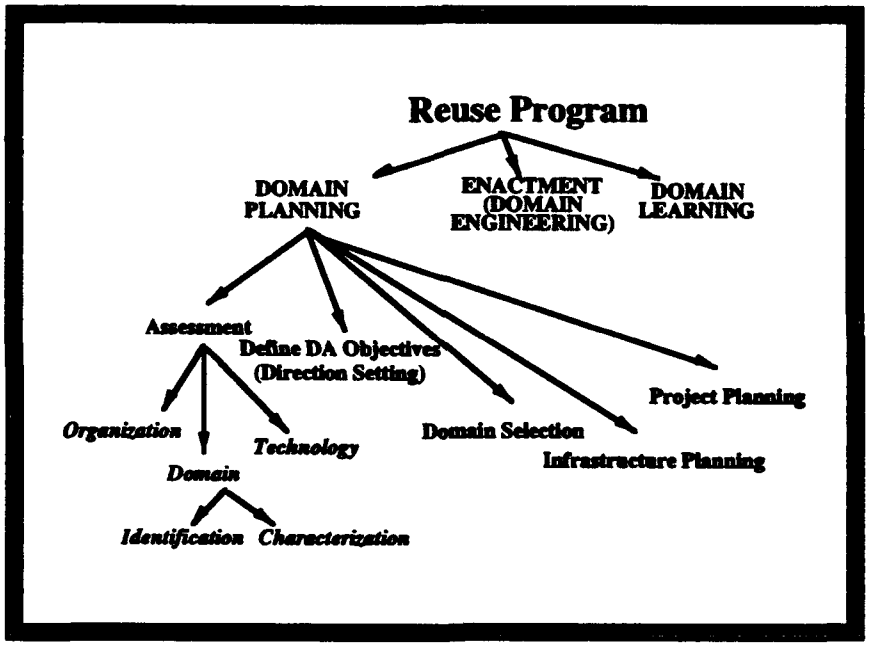


Figure J: ODM Domain Planning Process Tree

7.1.1 Reuse Assessment (ODM Aspects)

Purpose. Reuse Assessment processes help to establish an accurate picture of the current state of software engineering and reuse practice within the organization, in order to enable planners to set concrete and achievable objectives for the reuse program as a whole.

A subset of assessment activities are directly relevant to Asset Creation projects within the program scope, and specifically to domain modeling activities. These include assessment of the organization from the perspective of reuse potential and current capability, assessment of domains within the business area, and technology assessment, including the technical infrastructure in use within the organization and the software systems developed and/or maintained by the organization. While the immediate purpose for the assessment phase is

to assist in objective-setting, certain elements of the assessment will be inputs to later context-setting and modeling activities.

The rationale for the emphasis on organization, domain and technology assessment lies in ODM's concept of an *organization domain* as a shared abstraction intersecting, and integrating, organizational groups and structures, areas of expertise, and projects oriented around particular software products or systems. An organization domain will usually span several systems, may include only sub-portions of these systems, and will usually map across functional divisions within or even across organizations as well. The following subsections discuss these three aspects of assessment in more detail.

Assessment as Self-Assessment. In the ODM process as a whole, modeling is seen as both a way of acquiring and representing information, and a process that creates learning within the project team and the surrounding organization. Assessment is not merely preparation for modeling, but modeling in its own right, hence potentially an intervention. Consequently, the suggested emphasis is on a *self-assessment* process (possibly with outside facilitation) rather than an external auditing procedure. External audit-style assessments can yield useful information for the ODM process, but are counter to the style of the method.

Value of Outsider View. On the other hand, some outside facilitation of the assessment process can be valuable by providing the element of "insider-outsider" dialogue used in the domain modeling process itself. This is based on a view of assessment that emphasizes qualitative as well as (or in preference to) quantitative assessment techniques. Even assuming a time when the reuse field has access to well-supported, quantitative empirical research, a qualitative approach will still yield data otherwise unobtainable about the informal reuse practice and culture within the organization, and this culture is probably the most potent avenue for effective technology transfer.

7.1.1.1 Organization Assessment

7.1.1.1.1 Purpose

The ODM process model is a systematic approach to modeling and formalizing domain-specific expertise within a given technical community. Since ODM is intended to be tailorable to a broad spectrum of organizational contexts, the objectives and structure of the ODM project will depend to a large extent on fundamental differences in the ways organizations build and use software systems. Hence ODM cannot specify a single fixed sequence of processes. However, rather than simply offer a generic model that lacks detail specific to particular contexts, ODM includes organizational and domain assessment as an explicit step in the process model, and suggests heuristics for tailoring subsequent steps of the method based on this assessment phase. ODM explicitly recognizes a variety of pragmatic business contexts in which domain analysis and reuse efforts may be launched, and anticipates tailoring based on an organization's priorities (e.g., whether the project is driven primarily by available technology or by the needs of a particular line of business). This greatly increases the applicable scope of the modeling method, and facilitates comparison of diverse ODM projects via a common framework. The impact of this organizational typology is perhaps greatest in the domain selection process, which often involves a complex set of organizational and business decisions beyond assessing candidate domains' technical suitability for reuse.

7.1.1.1.2 Inputs

Since Organization Assessment is the initial process of Reuse Planning in the ODM context, inputs to this process include primarily general sources of information about the organization(s) within the program scope created through activities outside the ODM process. If the reuse program is a later cycle, then assessment workproducts from previous planning cycles may be available.

General sources of information about the organization might include: organization charts, business plans, marketing literature, annual reports, and informal knowledge of project team members. Informal knowledge should be used cautiously, however, particularly when the ODM project team has a different background and culture from prospective customer groups; e.g., when an R&D-oriented group will be doing modeling to create assets for an application engineering group.

Another valuable source of information would be the results of other organizational assessments that may have been performed or are in progress.

7.1.1.1.3 Resources

ODM Organization Typology

CFRP Idioms and Application Rules

7.1.1.1.4 Workproducts

Organization Software Engineering Profile

Business Area Stakeholders Model

Organization Reuse Readiness Assessment

Reuse Practices Survey

7.1.1.1.5 Process

The following activities are performed as part of this process.

7.1.1.1.5.1 Produce Organization Software Engineering Profile

Within the context of ODM, a major purpose of the assessment phase is to characterize the organization within a typology or business model, so that the DA process followed for the project can be tailored appropriately. This characterization, or software engineering profile for the organization, may consist of a number of components. The most critical for the purposes of domain modeling are discussed below.

Organization Sector Typology

The sector orientation of the enterprise undertaking the reuse program—whether government, government contractor, commercial software product developer, service organization, etc.—will significantly impact the objectives, and hence the processes and expected results, of the ODM project. For example, time to market may be a significant

business motivator for a commercial product developer, whereas reducing maintenance costs may be the major cost driver for a government maintenance organization primarily focused on post-deployment system support (PDSS). For a scientific, educational or non-profit institution, or for a commercial organization Issues such as classified or sensitive data, proprietary commercial assets vs. public data rights, and strategic models will also be significant factors.

Organization Software Engineering Typology

In its current form, ODM is oriented towards discovery and engineering of **software** domains. The role of software engineering within the organization therefore significantly impacts the potential approach to domain modeling. Specific points to address include the business model (e.g., a contract-, product-, or service-based organization), and life cycle phase(s) supported by the organization (e.g., development, prototyping, maintenance, IV&V). While related to the sector typology mentioned above, these aspects are somewhat independent; for example, government organizations could perform multiple roles.

Besides the software engineering business model and life cycle phases supported, it can also be useful to assess how software is used internally within the organization. For some large, enterprises (e.g., service-oriented companies such as commercial banks or airline companies) the amount of software developed to support internal business operations may itself warrant a domain modeling project.

When the organization is involved with software products or engineering applications, there is traditionally a split between the engineering and internal software groups. Yet much of the knowledge required for *organization domain* modeling may be embodied to some extent in these internal business support systems. There is therefore an advantage to viewing the use of software within the organization in a way that intentionally blurs these divisions along business lines, to facilitate restructuring along the lines of the relevant knowledge domains themselves. (In this respect, it can be instructive to compare the approach of ODM, and domain analysis in general, to *enterprise modeling* [???].)

7.1.1.1.5.2 Develop Business Area Stakeholders Model

As part of the Organization Assessment phase, a model of the various *stakeholders* within the business area is developed. Stakeholders include the most diverse group of people that can be considered to have an interest in the software systems and domains within the business area. They may include managers, software engineers, test developers, field technicians, technical support staff, documentation and training staff, end-users, value-added resellers, marketers, and even competing organizations. During the assessment process, those stakeholder groups relevant to the business area are identified and their interactions and relationships represented in a model. (The notation for the model is of less importance than the assessment process that makes the modeling team aware of the diversity of stakeholder perspectives available.)

Of particular importance from the standpoint of reuse and domain modeling will be stakeholder relations that correspond to Asset Creator/Asset Utilizer links, in CFRP terms. For example, software developers act as creators of the software systems employed by end-users in performing their operations. This relationship is not easily modeled as a simple producer-consumer relationship, or input-output relation, because end-users are not

receiving the software systems as inputs in this conventional sense. These relations are therefore usually not modeled in standard systems models of the end-user environment in which the systems will run, nor in business models of the development environment. Yet it is precisely across these boundaries that the most important reuse-based analysis may take place.

In general, the ODM method assumes that each business area, and each domain within a business area, may have a unique configuration of these stakeholder relations to be modeled. It is the unique aspects of each business area, and each domain, that will be of continued importance throughout the ODM project cycle. Later, the Business Area Stakeholder Model will be focused down to a Domain Stakeholder Model (after selection of the domain of focus). Eventually, each stakeholder context in this model will become a possible site for the binding of various features in the domain model.

Example: consider a manufacturer of software-intensive peripheral equipment such as laser printers. The business area/domain stakeholder model may include software developers, integration testers, field technicians from the producer organization, customer system managers who might install new software upgrades, third-party software vendors who might write driver software to make use of the peripheral's software, and, of course, end-users of the equipment.

7.1.1.1.5.3 Survey Reuse Practices

Before proceeding with more formal assessment of reuse technology transition issues for the organization (or the reuse program context) it can be helpful to perform a small-scale, informal survey of reuse practices within the organization. This can be done in a manner that does not suggest a formal assessment, with the implication of evaluation against a particular standard. What is of interest is the common practice within the development environment: how are resources shared, if at all? What informal repositories are used? How do people know where to find things? Who are the veterans, the experts in particular subject areas that serve as informal "knowledge bases" for the developers? What success and/or war stories about reuse efforts are part of the culture of the development group?

7.1.1.1.5.4 Assess Organization Reuse Readiness

Reuse readiness is an umbrella term referring to several parallel aspects: readiness in terms of concepts, capabilities and commitment. These are fairly independent aspects. One group may actually perform at a high level in terms of reuse, yet be relatively unaware of what they are doing in terms of formal processes. Another group may be aware of terminology and concepts in the domain, yet be far from practicing reuse themselves. The informal survey provides a good foundation for this assessment step. This is a primary context in which to consider the use of *ethnographic* data collection techniques, involving direct personal interviews, qualitative, open-ended questions, and possible interactions in interview settings among various individuals in the organization.

Concepts

Readiness in terms of reuse concepts refers to the exposure of the stakeholders to reuse concepts and methods, and also, to some extent, prevalent beliefs about reuse held by the group. For example, a developer group doing real-time applications may believe that attempts to reuse software will always result in inefficient code. This belief must be

identified and understood by the ODM project team in order to arrive at objectives and measurable success criteria that will speak to the appropriate concerns.

Capabilities

Capabilities of the organization with regard to reuse technology and practice can be assessed with a combination of the ethnographic techniques mentioned above and more formal assessment instruments, such as the SEI's Capability Maturity Model (CMM) [???] or the Software Productivity Consortium's Reuse Capability Model (RCM) [???]. This information will impact later choice of reuse technology, infrastructure planning, and asset implementation strategies.

Commitment

Commitment within a given organizational setting can be considered in terms of motivation, core values for the organization, and perceived risks and disincentives to reuse.

Motivation

Among other potential motivators for domain modeling efforts within an organization, the possibility of marketing the information derived from domain analysis directly should be considered. For example, domain analysis results could be marketed as a comparative survey, strategic marketing plan, or competitive analysis within the domain; or as a database or reference guide. For certain companies unaccustomed to this perspective, the shift from being product or service providers to information and knowledge providers may have profound repercussions.

Values

Unlike some other engineering techniques, motivation to reuse, or to design for reuse, at the individual engineer's level is often intimately connected with core professional values and even aesthetics. The reusable solution is often viewed as the "elegant general solution" that time constraints and pragmatics prevent from being realized. It is important to assess the dominant cultural and professional values in which the reuse effort is to be initiated. This will influence asset management and promotion strategies at the back end of the Asset Creation process; however, it will also influence the domain modeling project quite directly, as buy-in and commitment from domain stakeholders at all levels is essential to domain information gathering and validation activities.

Risks (Perceived and Actual)

It is important to understand the risks to a reuse-based approach that may be particularl to each business environment in which a reuse program is initiated. Results of the reuse practices survey will be instrumental in assessing what various stakeholders' concerns are, as well as technical areas of risk that may not be expressed by stakeholders but are evident to the project team. These risks should not be too quickly dismissed as "resistance" on the part of developers, since there will often be specific technical issues to be addressed. These issues should guide the formulation of ODM Project objectives, domain selection criteria, and choices about what kinds of metrics to maintain for the project.

7.1.1.1.6 Sequencing

Interaction with Wider Assessment Activities. Since activities described here are a subset of Reuse Planning activities as a whole, further assessment activities may be required for planning the Asset Management or Utilization projects within the program scope. Assessment activities required for planning the reuse program as a whole may overlap to a greater or lesser extent with corporate-wide assessment initiatives and needs, such as software development maturity assessment using instruments such as the SEI's Capability Maturity Model (CMM) [???], software productivity assessments [HP metrics ref???], or organizational assessments for the purposes of strategic planning, quality programs or line-of-business specific initiatives.

In beginning any assessment for the purposes of planning the ODM project, therefore, it is important to become aware of results from previous, ongoing or planned assessment activities within the organization. [Link to Foil #88, V2 ???] These results may be directly useful to the ODM project; awareness of other assessment initiatives may allow ODM results to be structured in a way more readily useful to groups outside the context of the reuse program, thus aligning reuse program and ODM project assessment activities and results within the larger organization.

7.1.1.2 Domain Assessment

7.1.1.2.1 Purpose

The purpose of Domain Identification is to generate a list of candidate domains, or *domains of interest*, within the business area scope of the reuse program. The ODM project will select one or more of these domains of interest to be the focus of planned domain modeling tasks. However, there can be other uses for the Candidate Domains List besides input to the selection process.

7.1.1.2.2 Inputs

Inputs to Domain Assessment may include the following:

Organization Assessment results, particularly statements about strategic lines of business or areas of expertise for the organization.

Inventory of software systems created and/or maintained by the organization. If possible these would be characterized by functional area(s) supported for each system.

Access to personnel knowledgeable about organization capabilities and strategic areas of interest.

7.1.1.2.3 Resources

ODM Candidate Domain Characterization Checklist

7.1.1.2.4 Workproducts

Domain Identification Criteria

Candidate Domains List

Domains Portfolio: This workproduct contains information generated during Domain Assessment that cannot easily be associated with only one domain within the business area. It should be developed with the assumption that future domain modeling projects initiated within this business area will be able to use the portfolio as a resource, and continue to evolve it. It potentially could include any or all of the following workproducts.

Business Area Information Sources Catalog : includes information sources for the business area as a whole, such as experts with knowledge across several domains, or high-level survey documents.

Domain Characterization Criteria : represents the tailoring of the general Domain Characterization Criteria Checklist for organization-specific features of interest. This would include input from analyses of an organization's core competencies, strategic domains of competitive interest, etc., or other criteria that would be part of the baseline characterization process for the organization for any domain.

Business Area Lexicon: represents terminology applicable across domains within the business area. This may be a source document from which candidate domain names are drawn.

Candidate Domain Characterization: includes the characterization information for candidate domains

Business Area Domain Interconnection Model: an initial version of this model that includes a cursory characterization of relations between domains within the Candidate Domains List. This model is not centered around a single domain of focus, and does not include the qualitative characterizations of the Domain Interconnection Model generated during Domain Scoping activities.

7.1.1.2.5 Process

The following paragraphs outline some suggested activities within the Domain Assessment process.

7.1.1.2.5.1 Articulate community's definition of "domain"

Before generating the list of candidate domains, it is helpful to capture what experts in the business area think a "domain" is. This definition establishes criteria to apply to candidate domains as they are identified. The "definition of a domain" itself is documented in the Domain Identification Criteria List.

To clarify: unlike the later Domain Definition stage, this step documents what experts consider to be "a domain" rather than "their domain". That is: how is a domain different from a system? What kinds of non-standard systems might exist in this software engineering context that are perceived as domains (scopes of applicability for engineering that are not defined by a single contract, project or product).

7.1.1.2.5.2 Domain Name Terminology

The technical community for each business area may share a unique set of terms for classes or families of systems. These terms themselves can be documented in a Domains Lexicon

that becomes part of the Domains Portfolio for the organization. This lexicon cannot be associated with only one of the resulting domains of interest, since it includes the language that allow experts to distinguish systems that are in different domains. It will therefore probably be best maintained as a separate lexicon. The Candidate Domain List is refined from these terms.

7.1.1.2.5.3 Domain Identification Criteria

The ODM team may decide to constrain their definition of what a domain is in a manner that filters the candidate list to a more uniform degree in terms of size, complexity and/or principle of organization. This will provide some advantages; for example, the characterization process will be simplified by having domains that are more consistent in nature.

However, attempting to normalize the domain descriptions too rapidly can also pose risks. One risk is that modelers will too readily adopt intuitive categories already used within the business area; this may conspire against the discovery of innovative kinds of domains, particularly novel horizontal domains.

By documenting the domain stakeholders' concepts of what constitute a domain, the domain modeling group's internal concepts, and the terminology for candidate domains drawn from an initial, high-level lexicon for the general business area of interest, these diverse perspectives can be taken into consideration in choosing a comparatively loose or tight filter on the domain identification process.

7.1.1.2.5.4 Domain Identification

Some domains have technical characteristics that appear promising for reuse; others appear higher-risk. The domain identification phase does **not** attempt to screen out domains that would not be good candidates for application of reuse technology. (This screening process will take place in steps leading to domain selection.) Instead, the object is to make explicit how experienced people within the general business area tend to cluster sets of systems into categories or work products.

If the ODM team includes experts in the general reuse context, these experts should document the terminology they bring to the modeling process. In effect, the approach here is to create prototype versions of the workproducts "off the top of one's head", to document obvious or intuitive ways in which these experts would structure the reuse context into discrete domains. It is **not** necessary to reconcile these viewpoints at this stage.

Candidate domains will not necessarily be disjoint; they may overlap and even enclose one another. The principles by which systems are clustered into various domains may differ widely from case to case. The level of granularity for different candidate systems may also vary widely. This is not an indication that the process has gone wrong. Since the objective is to find a suitable domain (a scope of applicability for reuse) each one of these domain definitions may prove a potentially useful basis for reuse engineering.

7.1.1.2.5.5 Domain Characterization

Candidate domains are characterized according to criteria also specified by the group. These criteria may overlap to a great extent those considered for domain selection, and it is

possible to see this activity simply as a precursor to the selection process. In this regard, it is most appropriate to apply general criteria

7.1.1.2.6 Sequencing

Reuse Planning and assessment activities should be performed with the expectation that domain boundaries will shift as a natural part of the analysis process. Stabilization of boundaries across and between domains is resolved as part of the Reuse Learning phase.

7.1.1.2.7 Guidelines

There are situations where no coherent line of business is defined for the context of the reuse program. Examples are DA projects initiated in an educational or training context, where individuals from different groups within the organization, or from different organizations, may be drawn together to undertake a DA effort. Another example would be a DA project initiated in a corporate-wide R&D setting, where there are no pre-defined ties to specific divisions or lines of business within the organization. In such situations, domain identification as a prelude to domain selection may appear to be a somewhat arbitrary exercise. Since the value of the Domains Portfolio is debatable in these contexts, the team should not devote excessive time to this task.

7.1.1.3 Technology Assessment

7.1.1.3.1 Purpose

Technology-neutrality is claimed as one of the desired features of ODM. Nevertheless, constraints on technology selection are a *legitimate* aspect of the organization context for the ODM project. These constraints may be negative (e.g., a technology may be inapplicable because of cost and resource constraints or technology transfer obstacles) or opportunistic (a particular technology may be almost a required aspect of any domain scope chosen).

7.1.1.3.2 Inputs

Inputs to Technology Assessment may include the following:

Organization and Domain Assessment results (as they become available);

Inventories of available hardware and software within the organization, procurement plans and technology surveys conducted by the organization;

Information about any technology developed in-house, particularly technology which the asset implementors may be constrained or encouraged to use;

Access to personnel knowledgeable about technology trends or the history of past technology transition efforts (both successes and failures) within the organization.

7.1.1.3.3 Workproducts

Enterprise Software Inventory: This inventory describes the applications or software systems that are within the scope of interest for the organization.

Reusability Analysis Report (???): This report details the constraints on technology selection for the ODM process, and later asset implementation, based on available technology and current technology usage within the program scope.

7.1.1.3.4 Process

Technology assessment views technology within the organization (alternatively, the scope of the reuse program) from several viewpoints. A survey of the software systems that form the available legacy can be undertaken, if the set of systems can be reasonably constrained before a specific domain is selected.

Technology forecasting addresses the exploration of long-term technology issues that might affect either the business area or the reuse technology being considered part of the infrastructure for the reuse program.

Another aspect of assessment is to survey the organizational communication infrastructure, including internal electronic mail systems, library groups, corporate communications and training and education departments. These resources will be useful during descriptive modeling as a way to broadcast requests for domain information; later, once an asset base is established, these same organizational functions will be instrumental in evolving a workable technology transfer plan.

7.1.2 Direction-Setting (ODM Aspects)

Direction-setting activities use the results of the Assessment phase (described above) in order to identify goals, constraints, perceived risks and specific objectives for the domain modeling effort.

7.1.2.1 Establish Domain Modeling Objectives

7.1.2.1.1 Purpose

The ODM method recognizes that DA is performed in the context of specific organizational objectives that impact the domain selected, modeling techniques applied, and reuse technologies chosen for implementation. Objectives are defined initially in light of overall reuse and general business objectives for the organization,

For example, DA could help define standard specification models supporting domain-specific completeness and consistency checking at the requirements level. A maintenance organization might want to reduce the maintained code base's size and maintenance burden. An organization that considers its depth of knowledge in a given domain as a key strategic asset might employ DA as a means of codifying and documenting veteran engineers' expertise.

7.1.2.1.2 Inputs

Inputs to the Direction-Setting activities may include the following:

Assessment phase results;

Strategic vision, goals, and/or objectives documents for the organization, including general business goals, software engineering goals, reuse-specific goals or goals characterized by domain-specific results.

Feedback from higher-level management and potential customer groups about expectations and perceived risks for the domain engineering project.

7.1.2.1.3 Workproducts

Domain Analysis Objectives Statement

7.1.2.1.4 Process

The following paragraphs outline some suggested steps for establishing the overall objectives for the domain modeling project.

7.1.2.1.4.1 Establish DA Constraints

Results of the assessment process can be used to determine constraints on the DA process, such as restrictions on the decisions and strategies of the DA team imposed by organizational and/or market conditions beyond their project scope. Constraints may include implicit (rather than formally documented) expectations about project results, perceived risks, and other factors that could compromise the success of technically sound results. For example, there may be a pragmatic constraint that the domain engineering effort cannot have any impact on the schedule or resources available for an application engineering project of high priority. This constraint, while not a specific objective, will nevertheless significantly shape viable objectives and strategies for the project.

In principle, identifying such constraints could be considered part of assessment activity. In practice, the task of identifying constraints will often be closely intertwined with direction-setting activity, since previously unconsidered or implicit constraints may be identified as possible objectives are evaluated and rejected.

7.1.2.1.4.2 Identify Objectives Context for Project

The objectives for the project must be defined with respect to the organizational context for the project. This includes objectives for the reuse program as a whole (i.e., including objectives for Asset Management and Asset Utilization projects as well as the Asset Creation activities within which the ODM process is performed). It may also include broade organizational objectives.

Two different hierarchical models for objectives context may be useful in this activity. The first distinguishes levels of objectives starting from general business objectives for the organization, filtered down to specific software engineering objectives, filtered in turn to specific objectives for software reuse, filtered yet again to specific objectives for Asset Creation and for the domain modeling project in particular. This same hierarchy will be reflected further in the ODM process, as general ODM project objectives will propagate to specific domain selection criteria and decisions about what kinds of models to build and what kinds of assets to develop for the domain.

A second hierarchical view places objectives at a more detailed level of abstraction than general vision, mission, goals, and strategy. In this framework, objectives usually refer to concrete measurable results that are in accord with more general goals and strategic principles. Specific objectives may then be further decomposed into sub-objectives, success criteria, relevant metrics, etc.

7.1.2.1.4.3 Brainstorm Concrete Project Objectives

In establishing Domain Modeling objectives, it can be useful to brainstorm potential end uses for the domain asset base. The value of this activity is to create a concrete vision in the team's awareness of real design goals and real users. The danger in this activity is to settle prematurely on specific implementation strategies as mentioned above, such as envisioning a library of components, an open architecture, or an application generator.

A good antidote to this latter tendency is to brainstorm alternative implementation strategies for any given project objective, and to allow these alternatives to remain as open possibilities rather than attempting to get closure on a decision during the objective-setting phase. In general, it is easier to avoid fixing on specific solutions by generating multiple alternatives, rather than by trying to avoid thinking of implementation possibilities altogether.

7.1.2.1.4.4 Refine Objectives

Based on the objectives context established previously, the most viable objectives for the project are refined and documented. Linkage from each specific project objective to the constraints and context objectives are also documented. Based on this linkage, each objective can be evaluated to verify that it is consistent with the contextual constraints and objectives. It is likely that additional objectives will be established that are not merely reflections of external requirements on the project. The set of objectives as a whole can be evaluated to determine whether it constitutes a sufficient guarantee of success for the project; that is, if all objectives are satisfied, will the project be considered a success? Redundant or superfluous objectives can also be detected during this validation stage.

The objectives should be documented clearly and concisely, in language that will readily communicate to new project members over the lifetime of the project. The objectives also serve as a statement of expectations for upper management. In reviews and presentations to validate the objectives with all stakeholders for the project, it is important to verify not only acceptance of each explicitly documented objectives, but also sign-off on the **absence** of certain objectives as well.

7.1.2.1.4.5 Establish Success Indicators, Metrics, Evaluation Criteria

Once a detailed set of specific objectives has been developed, refined and validated, the objectives can be further decomposed into specific success indicators, metrics and evaluation criteria. These will determine aspects of the overall project plan, such as what kind of metrics data needs to be collected to demonstrate that particular objectives have been met, and when review and evaluation activities need to be scheduled to ensure that progress with respect to objectives is monitored at reasonable intervals.

7.1.2.1.5 Sequencing

Objectives are established after the assessment phase in ODM process (as in the CFRP generally), in part so that assessment of the organization's reuse capability and overall business objectives can be reflected in the specific project objectives. Objectives are preferably established before domain selection, except in situations where choice of domain is highly constrained by the business context.

While an idealized process model would suggest that these objectives must be defined in the sequence as prescribed, it can also be the case, especially for teams attempting domain modeling for the first time, that the need for explicit setting of project objectives will not really become evident until later in the process. For a team unaccustomed to a heavy process orientation, it may be advisable begin domain selection with objectives stated relatively informally, with clear intent to return to general project objectives when a conflict is reached. Conflicts will then naturally arise if there is an insufficient basis for selecting among candidate domains. The domain selection process should be the earliest phase where the lack of explicit objectives causes this breakdown. As long as the domain selection process is not closed, backtracking to a more formal objective-setting process at this point should not cause undue loss of quality to project results.

7.1.2.1.6 Guidelines

Scoping the Domain Modeling Objectives

Domain Modeling objectives cover the full range of modeling activities. A typical mistake is to confuse these objectives with domain selection criteria, which are derived in part from these objectives but are defined at a later stage.

Domain modeling objectives should be stated in a form not unduly constraining technology choice (e.g., building a library vs. an application generator is an implementation choice, not an objective).

7.1.2.1.7 Issues

Complexity of Reuse Management

The CFRP enables recursive levels of reuse program and project planning, each potentially with their own direction-setting process. In practice, it appears that when these levels of planning get too close, the distinctions between objectives at different levels get too subtle for project teams to negotiate usefully. This should be considered when doing overall project planning; if a scope of planning does not easily lend itself to clear and distinct objectives, it may be an unnecessary level of management to the project structure.

In many organizations, the higher-level objectives required as input to guarantee a truly business-aligned domain modeling process will not be readily available. In these cases, seemingly low-level technical issues will tend to propagate up to wider management concerns. The project manager should track the direction-setting activity closely enough to intercept these questions and act as liaison to get them resolved for the modeling team. While this is true of any technical development, it does seem to be a characteristic of domain modeling projects that they act as magnets for these broader strategic concerns (e.g., questions of organization core competencies and product planning strategies).

7.1.3 Domain Selection

The CFRP Domain Selection process category applies to the reuse program scope as a whole, which may contain multiple Asset Creation, Management, and Utilization projects. ODM assumes the context of a single Asset Creation project. This single project may, in principle, choose a scope that includes several domains, as they are defined via Domain Identification and Characterization for the Business Area. These composite domain scopes may result from the composition of smaller domain scopes as part of Domain Identification, or through the tactical selection of multiple, relatively unrelated domains for a multi-team effort. (The line between the latter and separate Asset Creation projects, in a CFRP sense, could become somewhat blurred.) The description below will encompass the selection process for a single Asset Creation project.

7.1.3.1 Select Domain for ODM Project

Choice of DA objectives and selection of the domain will influence the specific DA method or methods selected for the effort.

7.1.3.1.1 Purpose

The Candidate Domains List produced during Domain Identification and Characterization will be created initially for an organization, then change only slowly (e.g., if workers develop expertise in new areas, or if existing areas are partitioned into new domains). Domain Selection, in contrast, involves a decision process that occurs for each domain analysis project undertaken. Domain selection criteria will be influenced by overall reuse and domain analysis objectives, and by the near-term constraints such as availability of domain experts, the current life cycle phase of likely recipient application projects, etc.

7.1.3.1.2 Inputs

Inputs to the Domain Selection process may include the following:

ODM Assessment Phase results;

ODM Project Objectives;

Candidate Domains List;

Candidate Domains Characterization;

Domains Portfolio;

Possible inputs from parallel domain modeling efforts, within or outside of the program scope.

7.1.3.1.3 Resources

ODM Domain Selection Criteria List

7.1.3.1.4 Workproducts

Domain Selection Criteria List

Candidate Domain Characterization

Domain Selection Report

7.1.3.1.5 Process

7.1.3.1.5.1 Define Domain Selection Criteria

Criteria for domain selection are defined before the selection process itself. The main reason for this step is as process and rationale capture. In addition, the step serves to integrate general knowledge about selection criteria for reuse with context-specific objectives and constraints on the selection. The criteria, therefore, should reflect as much as possible the particulars of the organizational context in which the selection is being made.

Documenting criteria also helps to reveal implicit or controversial constraints on the selection process. It can be particularly useful in providing a requirements structure for sign-off from upper management, without leaving the selection process entirely under management control.

7.1.3.1.5.2 Characterize Candidate Domains

Potential domains are characterized and prioritized with respect to criteria relevant to the organization. This distinguishes this step from the characterization performed during Domain Assessment, which in principle could be performed before the definition of specific objectives for the ODM project.

7.1.3.1.5.3 Choose the domain and document rationale

The selection is performed and the rationale documented. Although the candidate characterization activity may imply a rational, priority-driven selection process, the final choice may be made on the basis of unpredictable factors. An effort should be made to document the real process and real concerns that motivated the decision. One target audience for the rationale document will be planners of subsequent domain modeling projects within the same business area.

It is important to obtain buy-in from the entire project team on the domain selected. While not all decisions in the ODM process can be made on a collaborative, consensus basis, choice of domain is a decision that will have pervasive impact on the entire project; any disaffected parties with regard to this decision may be difficult to engage later on in the project.

7.1.3.1.6 Sequencing

Definition of selection criteria may need to be reverse engineered after a de facto domain selection has been made. It is still worth going back and documenting the criteria separately from the decision itself. Often, conflicts or breakdowns in getting consensus will provide insight about missing or undocumented criteria. (See Example.)

7.1.3.1.7 Guidelines

General criteria might include: the stability of the underlying technology within the domain; the maturity of the domain (number of systems implemented and length of time

fielded); the estimated commonality of functions across applications; or the degree to which performance constraints may inhibit reuse of components with excess functionality.

7.1.3.1.8 Issues

Some characteristics may be difficult to interpret as either positive or negative indicators for the selection process.

The definition of selection criteria and the selection process itself can be iterative. An initial set of criteria may seem satisfactory. However, when a tentative domain choice is considered that satisfies all current criteria, yet clearly feels intuitively wrong or fails to get consensus from the modeling team or supporting management, this breakdown may usefully reveal additional constraints on the selection process that have not been explicitly documented. No criterion is "wrong" if it reflects a real contextual constraint. A common fallacy is to ignore situation-specific constraints that are felt to obscure the purity of the reuse endeavor. Criteria can conflict and even be mutually exclusive. The prioritization step can assign relevant weights to the different considerations.

7.1.3.1.9 Example

The STARS/Army Demonstration project discovered the constraint that the domain selected should cover functionality on the main computer software of the systems of focus, rather than firmware-embedded functionality in peripheral equipment. This criterion was clearly linked to their project context, and is not a general criterion for domain selection. For example, within some Hewlett Packard application groups, reuse within firmware domains is entirely appropriate to the business area. Documenting this context-specific constraint was therefore important for the demonstration project as well as for others who would want to interpret their results and apply them elsewhere.

7.1.4 Infrastructure Planning

As with other aspects, infrastructure planning for the program scope as a whole will span the Asset Creation, Management and Utilization projects. This emphasizes the fact that domain modeling methods and technologies cannot be selected independently of the larger program context in which they will be used. The section below addresses the issues in planning the infrastructure for the ODM project specifically within this wider context.

7.1.4.1 Plan Domain Modeling Infrastructure

7.1.4.1.1 Purpose

The notion of infrastructure is fundamental to both the CFRP and ODM frameworks. Applying ODM Stakeholder Modeling concepts to the use of ODM itself on a project, we can distinguish several potential layers within which distinct learning feedback loops and/or producer-consumer information flows might be discerned. This is illustrated in Figure K, below.

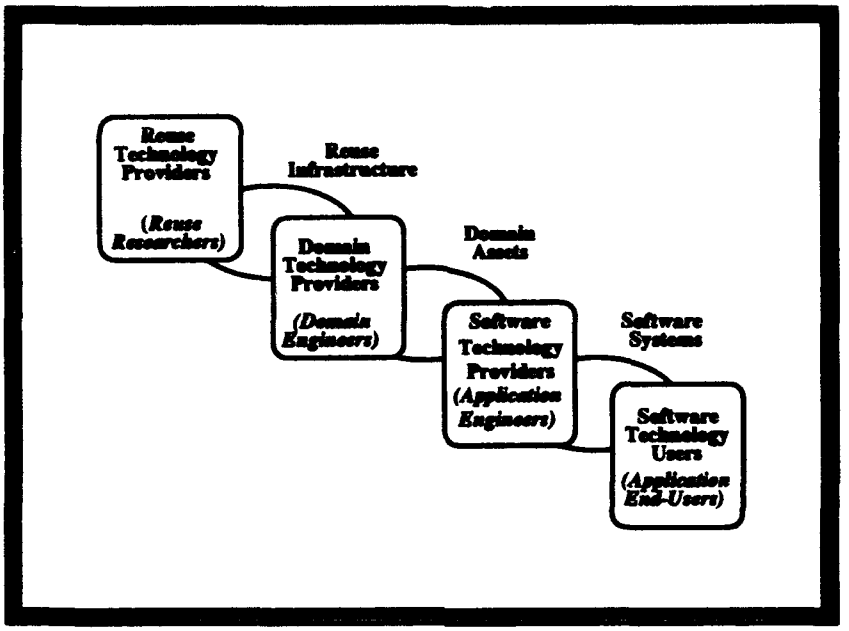


Figure K: A High-Level Domain Stakeholder Model

At the end-point of the technology development process modeled in any program scope are *end-users* of the applications being developed. These end-users are the customers of the *application developers* who build the software systems; these developers make use of the developer organization's infrastructure for software development. In the current state of software practice, such environments are usually based on general-purpose software development tools, models and methods. The *domain developer* can be viewed as an engineer that tailors and/or adapts general software engineering infrastructure to the needs of application developers in specific domains; this may include encapsulating and reengineering domain-specific components for reuse or formalizing methods and processes developed within the domain to provide more consistent automated support. The software components, domain-specific tools and generators, models and other documentation of domain-specific knowledge produced or codified by the domain developers is maintained in an asset base that, in effect, becomes part of the infrastructure provided to the application developers in a reuse-supportive engineering life

From the standpoint of the domain developer, however, a different infrastructure is required. This is the infrastructure supporting asset creation processes themselves, including domain modeling (using ODM or any other methods). Whereas components in a software library are part of the asset base, the library toolset and mechanism itself can be considered part of the infrastructure (for the asset creator). As reuse technologies and methods mature, this asset creation infrastructure will become itself an instantiation of a supported set of capabilities evolved and maintained by reuse technology and methods providers.

Given this contextual definition of infrastructure, the purpose of this planning step within the ODM framework can be described as selecting, adapting, tailoring, or developing the asset creation infrastructure, specifically the support needed for the domain modeling tasks of the project. One feature of ODM is that commitments to various technologies are made

in a phased way; thus, asset implementation technology is selected on the basis of results from descriptive and prescriptive modeling stages. During initial planning, therefore, the focus is on the infrastructure required for descriptive modeling.

7.1.4.1.2 Inputs

Inputs to the Plan Domain Modeling Infrastructure activity may include the following:

Organization and Technology Assessment Results;

ODM Project Objectives;

Domain Selection Report.

7.1.4.1.3 Workproducts

ODM Project Handbook: This handbook becomes the repository for shared information among project members about methods, processes, policies, and technologies.

ODM Project Infrastructure Plan: Identifies constraints, specific criteria, and ranges of options available for technical, organizational and educational infrastructure for the project.

7.1.4.1.4 Process

The CFRP distinguishes three broad categories of infrastructure issues: technical, organizational, and educational infrastructure. Each of these are relevant for the subset of the infrastructure focused on domain modeling support, and will be described below.

7.1.4.1.4.1 Plan Organization Infrastructure

Organization infrastructure includes institutional support for the reuse program, such as steering committees, technical interchange forums, project review structures. The purpose is not to introduce needless managerial overhead, but to verify that organizational structures have been put in place to enable an effective domain modeling effort.

While this planning step would be necessary in any approach to DA, the need is particularly acute in ODM because of the notion of an organization domain as one where knowledge about (and interest in) the domain crosses divisional and functional boundaries within the organization. This requires a careful handling of issues such as project ownership, accountability, and access to diverse groups up front. In principle, since availability and accessibility of domain expertise is a key criterion for domain selection, the domain selection process may be affected by the outcome of this planning process. During this step, project planners may select and tailor specific DA methods, and specify requirements for DA tools. This step includes the selection of particular DA methods and representations to apply.

7.1.4.1.4.2 Plan Technical Infrastructure

This activity identifies global constraints on technology infrastructure (e.g., tools, environments, and methods). Since multiple models will be developed, for artifacts from different life cycle phases, each project may select from a spectrum of techniques to apply;

however, there is a set of possibilities (or *repertoire)* that constrains the range of these techniques.

7.1.4.1.4.3 Plan Education/Training Infrastructure

Educational and training infrastructure refers to mechanisms in place for introducing new tools, methods and policies within the project team. These issues closely overlap the other aspects of infrastructure described above. New technology requires training and method support; such training must be supported at the organizational level. For the foreseeable future, domain analysis projects in most organizations will simultaneously build to produce concrete results and to educate the project team and the surrounding organization (and possibly to provide feedback to reuse technology providers as well). Interfaces from the project team to the organization's education and training groups and to outside recipients of project lessons learned need to be planned and/or monitored carefully, since the various learning loops superimposed on the modeling tasks can easily overwhelm available resources .

7.1.5 Reuse Project Planning

7.1.5.1 ODM Project Planning

7.1.5.1.1 Purpose

ODM Project Planning is an ongoing activity that oversees the domain modeling effort as a whole. Initially, it serves to establish and consolidate the team and tailor the general ODM process model into a specific process plan for the project. This process also includes managing certain external interfaces.

7.1.5.1.2 Inputs

Inputs to the ODM Project Planning activity may include the following:

ODM Project Objectives

ODM Domain Selection Report

ODM Infrastructure Plan

Assessment Results

7.1.5.1.3 Workproducts

ODM Project Plan

ODM Project Process Model

Domain Dossier Structure

Domain Model Validation Plan

Schedules, Milestones, Resource Plans

7.1.5.1.4 Process

7.1.5.1.4.1 Form ODM Team

In forming the ODM team, there are a number of criteria to consider.

Team Diversity. Diversity in the team structure will complicate the management task; if successful, however, diverwsi[illegible]ich the quality of domain models by providing validation from multiple persp[illegible]. Circulation/rotationof personnel between domain and application engineering teams, particularly for projects of some duration, is a good strategy for strengthening the connection of the modeling team to the applications groups.

Level of Domain Knowledge. Modelers' knowledge about the domain they study can range from expert to novice; each perspective can add value to the modeling effort. Certain processes within ODM will be best suited for the "naive", others for "expert" domain modeler. Furthermore, this distinction is not based on the simple case where the naive modeler must do all the steps of the expert modeler, plus others to come up to speed in the domain. On the contrary, for certain steps, modelers' prior knowledge of the domain should be documented to establish potential perceptual biases with which they approach the modeling task. Thus, there are advantages and disadvantages to both naive and expert modelers' perspectives.

The most effective ODM teams will therefore intentionally strive for diversity with this respect, and include a spectrum of familiarity with the domain, as well as those experienced in related or analogous domains who bring yet another viewpoint to the process.

Peripheral Experts. There will often be a significant culture clash between the applications developer environments (the customers for reusable assets) and the software engineering teams that perform domain modeling. This can present a considerable challenge if members of both comunities are to function together on the modeling team. It may be appropriate to identify certain domain experts as expert reviewers "of first resort" for the project team, while welcoming but not being dependent on more full-scale participation. Another alternative role is in creating and validating the prototype lexicon (described below in Section 7.2.3.2) which requires extensive domain knowledge but defers detailed modeling activities.

7.1.5.1.4.2 Define Project Process Model

A core element of the CFRP Reuse Management idiom is the concept of Reuse Enactment as intentional specification of a process and a commitment to follow that process or to document departures from it. Since ODM assumes a CFRP Reuse Management context for a domain modeling project, definition of the process to be followed for domain modeling is integral to project planning.

While process modeling has validity for any project management, it has particular significance for ODM projects, for several reasons. First, relative to conventional development projects, there are more complex team interactions required for effective domain modeling, both within the ODM project team (because of the team modeling approach) and between the team and other projects, organizational entities and domains. Thus, a project small enough in terms of personnel to tolerate informal processes under normal circumstances may need greater formality in its approach to process management.

For ODM, large-scale projects will, by implication, require greater process skills on the part of participants and a greater degree of process support from the technical infrastructure.

Explicit processes are also inherent in ODM because the model assumes and even requires project-specific tailoring of workproducts and process models. Even when detailed case histories and example process plans are available as resources, the unique organizational, domain, and technology profile of a given reuse program will always require customization of the generic ODM process model. Furthermore, since subsequent cycles of a reuse program incorporate learning feedback from previous cycles, process models will evolve over time, even within a specific organizational context.

This requirement for project-specific tailoring does not imply that processes are always defined up front. Particularly when a team is undergoing a process for the first time, or when the methods themselves are in rapid evolution, it may be more appropriate and realistics to conduct a prototype version of a project phase first, then document the process actually followed and reflect on what appeared to be successful and/or problematic.

7.1.5.1.4.3 Define Project Interfaces

This activity centralizes issues concerning the interaction of the ODM project team with other teams and organizational entities.

Primary interactions to establish for the ODM project team are those that link Asset Creation activities with Asset Management and Asset Utilization activities within the reuse program scope. For example, if multiple, parallel domain engineering efforts were initiated within a given scope, it would be important to maximize the potential synergy between these two groups, while minimizing the potential chaotic impact of too many unofficial interfaces and cross-connections. Additional interfaces can be identified with: the parent organization (or organization context), or related domain engineering efforts within or outside the scope of the reuse program.

7.1.5.1.4.4 Obtaining Commitment

Domain modeling is and will remain a high-risk undertaking. Obtaining commitment, or buy-in, to a project plan is always a necessity; however, this commitment is particularly crucial to the formation of a domain modeling team with the dynamics necessary for successful results. For a domain engineering project, buy-in is necessary at all levels. For engineering staff to commit to the project, they must perceive a high level of management support. This will be equally true of asset utilization groups; however, in these cases, management support cannot translate into management fiat. Assets must be reused on their own merits; policy and incentives can remove disablers but cannot manufacture enablers unless technical solutions are acceptable to the engineers. Thus, skeptics can be valuable assets on the domain modeling project team; they should be cultivated and, if possible, brought into a formal reviewer's role. By surfacing potential conflicts and issues early in the process, they can act like the proverbial "canary in the mine shaft".

7.2 Descriptive Domain Modeling

This section describes the Descriptive Domain Modeling phase of ODM. This phase is a key differentiating feature of ODM, as it offers an alternative way of deriving models and architectures that are directly produced in other DA methods. Descriptive modeling

includes a context-setting, model evolution and model refinement phases that can be thought of as specialized versions of Plan, Enact and Learn processes. The overall structure of the Descriptive Modeling phase is illustrated in Figure L.

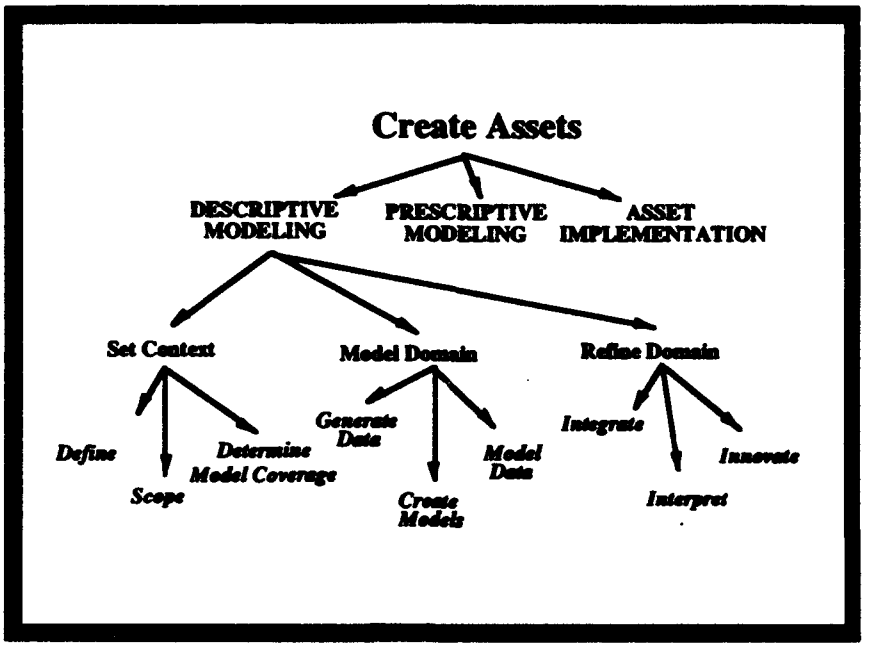


Figure L: ODM Descriptive Modelling Process Tree

Section 7.2.1 provides rationale for the descriptive modeling approach. Subsequent sections detail the specific processes within the Descriptive Modeling phase.

7.2.1 Rationale

The first major phase in the core domain modeling activity of ODM is the descriptive modeling phase. The domain modeler observes a discipline whereby concepts described in the descriptive model (or, later, features mentioned in the descriptive feature model) are justified by the precedent of occurring in a system or system requirement included in the domain exemplar set. Linkage with the exemplar set establishes a basis for scoping the features to be modeled.

The exemplar set can include both existing systems, systems currently under development, known requirements for new systems, and forecasts of potential market demand, derived from marketing analyses, strategic planning, etc. The emphasis in all cases is on providing concrete fixed points of reference to determine what should be in each domain model. The exemplars may also not include entire systems: for horizontal domains, we may be looking at a function of a system, either encapsulated as a discrete sub-component or a distributed subset of functionality. The term "exemplar" covers both system and sub-system level scopes. The concept of vertical, horizontal, encapsulated and distributed domains are depicted in Figures M and N below.

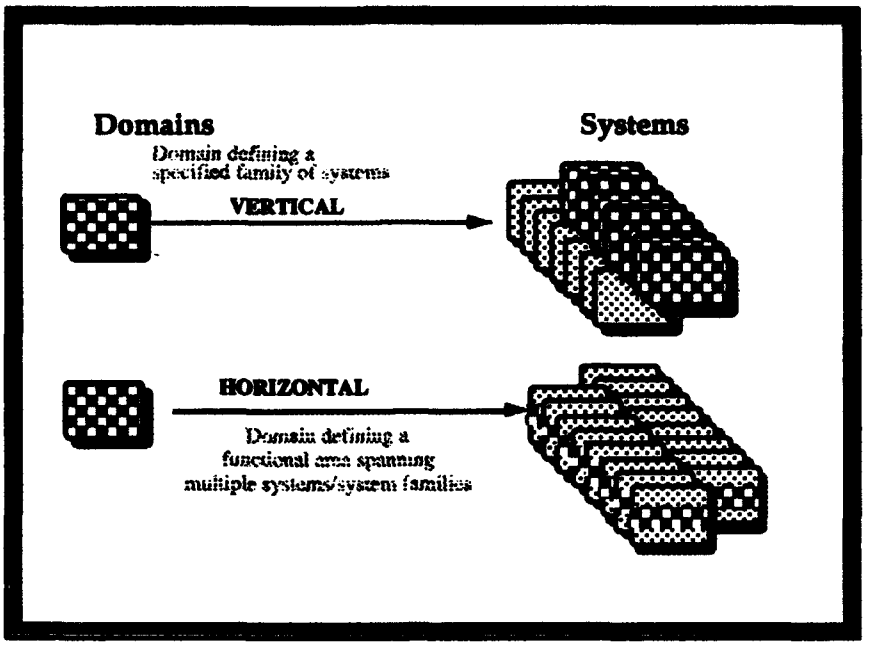


Figure M: Vertical Vs. Horizontal Domains

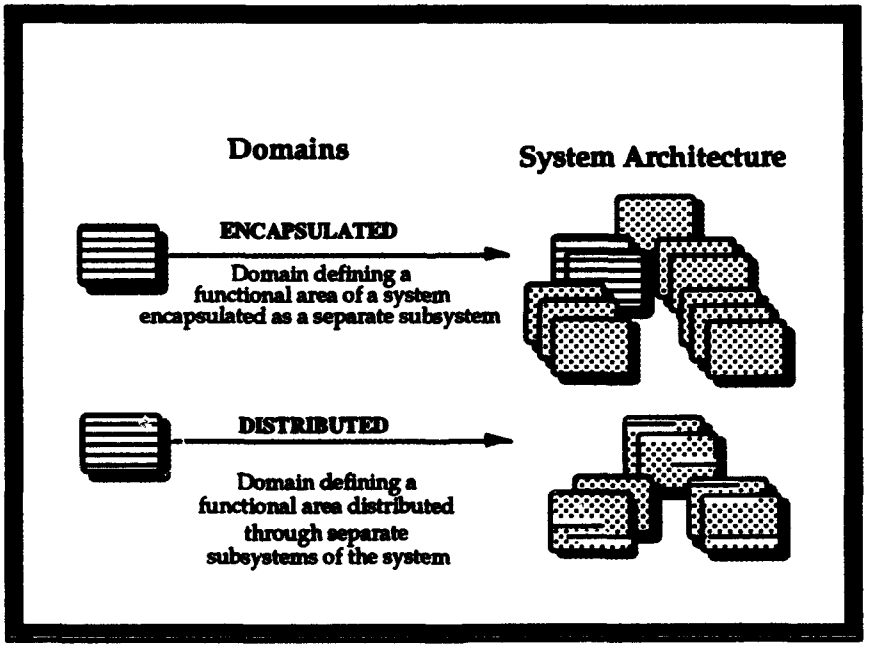


Figure N: Encapsulated vs. Distributed Horizontal Domains

Each exemplar system has related communities of *stakeholders* that have varying *interests* in, and various kinds of knowledge about, th tem and the domain-specific functionality. Given a set of different exemplars, any given akeholder will find certain differentiating features of interest with respect to their potential interaction with the system.

Example: In the outlining domain, an end-user may care about the ability of different outliners to include paragraph text associated with each outline heading level. An outline developer may care about the performance of a potential reusable component.

A descriptive domain model should have sufficient expressive power to characterize what a specific community of domain stakeholders consider to be significant commonality and variation in features across the specified exemplar set. By linking each concept included in the descriptive model to a *precedent* in at least one exemplar, the ODM method allows for traceability and validation of domain models on an empirical basis, and provides exit criteria for the descriptive phase of the modeling process.

Holding to the descriptive modeling discipline does not have to mean stifling innovation. In fact, a number of workproducts and steps in the ODM process are intended to spur innovative thinking about domain functionality, but in a managed process. When domain modeling yields innovative but unprecedented feature ideas (for example during the process of analyzing and probing analogy domains), these extra features are simply documented separately from the precedented features. Seemingly indispensable features for which there is no exemplar precedent may lead domain modelers to later expand the exemplar set, or can be modeled later as part of the Domain Innovation Model.

7.2.2 Context Setting (Plan)

This section describes the context-setting activities within Descriptive Modeling. These include defining the domain, scoping the domain and determining the modeling coverage for the domain. At the end of this phase, the informal definition and boundary for the domain has been formalized and the information collection task has been properly bounded.

7.2.2.1 Define Domain

7.2.2.1.1 Purpose

The first step in Descriptive Domain Modeling is to define more formally what is really entailed by the selected domain. During domain selection, the domain has usually been defined via an informal, tacit understanding shared among knowledgeable experts. This is not a sufficient basis for scoping the DA process. The primary purpose of the domain definition is to clarify what systems or system functionality is included in and what is excluded from the domain.

7.2.2.1.2 Inputs

Inputs to the Domain Definition process may include the following:

Material from the Domains Portfolio;

Domain characterization information from Domain Assessment phase;

The brief domain definition statement developed during domain selection.;

Access to domain informants, for data concerning domain terminology relevant to definition of the domain.

7.2.2.1.3 Workproducts

Domain Definition Statement

Domain Exemplar Set

Domain Genealogy Model

7.2.2.1.4 Process

The following paragraphs describe the key activities in the Domain Definition process. For this process, the activities are closely linked to the workproducts produced.

7.2.2.1.4.1 Domain Definition Statement (DDS)

The DDS is an *intensional* definition of the domain: i.e., rather than enumerate or define through examples, it provides a set of criteria for determining whether a given system falls within the domain or not.

At a minimum, the domain definition serves as a consensus document for the ODM team. In addition, the domain definition should allow experts in the application area to understand what specific scope is addressed within the domain of focus. If possible, it can also be desirable that the domain definition communicate to someone who is not an expert in the domain what the essential functionality provided by systems in the domain.

The domain definition may be used by domain planners who wish to understand the potential overlap between distinct domains. (This overlap would be more fully described in the Domain Interconnection Model, if both domains had been considered during the Domain Scoping process.) The DDS can also aid component developers and asset managers. Refined and updated throughout the prescriptive modeling phase to represent the actual scope of the final asset base, the Domain Definition should help potential utilizers determine whether the component they are seeking belongs within the scope of the domain. The definition should aid someone who is **not** an expert in the domain understand the primary function of systems or components in that domain, to determine whether the asset they seek (or offer) belongs within the scope of that domain. In addition, since Utilizers can offer workproducts as draft components to the Asset Manager, or request components that are not in the asset base, the Domain Definition serves as the basis for distinguishing assets absent from the asset base due to deliberate exclusion, as opposed to those absent because of gaps in coverage that could properly be addressed with a new asset.

7.2.2.1.4.2 Domain Exemplar Set (DES)

The *Domain Exemplar Set (DES)* is a list of candidate systems for the domain. The DES provides an *extensional* domain definition that complements the Domain Definition Statement. It defines the domain by identifying exemplar systems considered members of the domain.

An important purpose of the DES is to provide a basis for empirical validation of the DDS. All feature variants included in the DDS will be justified by reference to some precedent in an existing system or new system requirement in the DES. (Usually, but not always, these features will be drawn from the Domain Representative Set.)

Categories of Systems

The systems in the DES should be categorized roughly with regard to their "degree of membership" in the domain. Relevant categories include:

Core exemplars: These are systems considered central to the domain definition. They may include systems that are considered the front runner or established standard within the domain, especially in commercial software arena.

Borderline exemplars: These would include exemplars that raised issues or disagreement within the ODM team. Usually such borderline cases are symptomatic of a domain boundary issue which may be iteratively explored and resolved via adjustment of the DDS and the Domain Interconnection Model, and documented in a Domain Boundary Decision Report.

Counter-exemplars: These are systems that lie "just outside the border" of the domain. (Obviously, listing all counter-exemplars would be an unbounded task!) They are documented primarily so that outside users of the DA results can distinguish between systems that were not considered by the team, versus those considered and rejected for specific reasons. This could include systems recognized at the outset as being excluded by the domain definition, as well as systems that raised issues and discussion and were eventually allocated outside the domain. This category is somewhat complementary to the Borderline Exemplar category.

Span of systems

The DES should include, if possible:

- existing systems (where source workproducts are available for analysis)
- requirements for new or potential systems
- and "market" studies of general new areas of demand in the application domain.

Rationale: We want the exemplar set to span a broad diversity of systems, so that the descriptive model is as rich in variability as necessary given the scope of the domain.

The decision to model an older system does not constitute a commitment to either adopt the architecture or design of that system, or to design for reuse in a manner allowing for incorporation into that system. Reengineering may not be possible or even desirable for some legacy systems. Domain Genealogy information (defined below) will help provide the data to allow superseded or obsolete functionality to be flagged in older system versions.

Number of systems

The DES should contain as many systems as the ODM team has had a chance to examine and classify as belonging to the domain.

Placing a system in the DES does not constitute a plan to study it in detail. After characterizing exemplar systems and their inter-relationships via the Domain Genealogy Model, the set of exemplar systems to be studied intensively (the Domain Representative Set) will be selected.

7.2.2.1.4.3 Domain Genealogy Model

The *Domain Genealogy Model (DGM)* encapsulates data on the evolution and historical interrelationships and dependencies among systems within the domain. Documenting these historical relationships helps ensure selection of a representative exemplar set with sufficient spread and appropriate interpretation of commonalities across systems with qualitatively distinct genealogical relations.

Genealogical relations currently described in ODM include the following:

- *direct successor:* system A superseded system B, presumably adding new features, correcting errors, and observing varying constraints on upward and/or downward compatibility;
- *leveraged:* code (and/or design, tests, etc.) from system A was adapted for use in system B, possibly to implement a system for a new customer;
- *requirements reuse:* operational features of system A were adopted for system B (typical for software products of competitors who begin "feature wars"); and
- *independently developed:* systems were implemented in different contexts, and without reference to each other (e.g., applications developed in-house within separate organizations or divisions; or
- *competitively developed:* systems were developed for the same target environment (e.g., shadow projects, redundant systems development for reliability, competitive designs to published requirements, applications developed in educational settings).

The set of domain exemplars chosen for intensive study should include as good a cross-section of these relations as possible, with particular emphasis placed on independently developed systems; commonality observed across such systems is more likely to represent core functionality for the domain.

7.2.2.1.5 Sequencing

Domain Definition activities can be performed largely in parallel. For example, two independent groups can develop the intensional and extensional definitions, iteratively validating results against the results of the other group. The genealogical view of the domain may have been initiated as part of the Technology Assessment phase, and only needs to be focused on the systems of specific relevance to the domain. This additional information would usually be obtained through similar sources as those required to develop the Domain Exemplar Set.

7.2.2.2 Scope Domain

7.2.2.2.1 Purpose

The Domain Definition process establishes principles of inclusion and exclusion for the domain. In Domain Scoping, modelers revisit the various boundaries established around the domain and qualify these boundaries, assigning other domain names to bordering areas of functionality and classifying the types of relations that hold between the domain of focus

and these related domains. The distinction between definition and scoping can be viewed as inside-out vs. outside-in definition of the domain. Articulating appropriate domain boundaries is one of the most difficult tasks in DA, and occurs iteratively throughout the modeling process.

7.2.2.2.2 Inputs

Inputs to the Domain Scoping activity may include the following:

Domain Definition Statement

Domain Exemplar Set

Domain Genealogy Model

Domain Stakeholders Model

7.2.2.2.3 Workproducts

Domain Interconnection Model

Qualified Exemplar Set

Domain Boundary Decision Report

7.2.2.2.4 Process

The following paragraphs describe the activities in Domain Scoping. For this process, these activities are tied closely to the workproducts produced.,

7.2.2.2.4.1 Domain Interconnection Model

ODM uses a multi-faceted Domain Interconnection Model to record different kinds of relations between domains, including: operational relations; specialization and generalization relations; analogy relations; and development environment relations.

This model is quite different from a conventional systems model, or even a model of assets in an asset base. Relations in the Domain Interconnection Model are defined between domains, not systems or domain assets.

7.2.2.2.4.2 Qualified Domain Exemplar Set

The Domain Exemplar Set resulting from Domain Definition includes core, borderline and counter-exemplars for the domain; however, borderline and counter-exemplars are not further characterized in relation to the domain of focus. Just as the intentional and extensional definitions served as cross-validation during Domain Definition, the Domain Exemplar Set and Domain Interconnection Model can be used for cross-validation of the definition and scoping phases. The Qualified Domain Exemplar Set is an annotated and possibly extended version of the Domain Exemplar Set. Borderline and counter-exemplars are characterized in terms of related domains; where no related domain can be associated with the exemplar, the set of related domains is examined for completeness and

consistency. Related domains with no borderline or counter-exemplar may result in additional investigation of candidate systems.

7.2.2.2.4.3 Domain Boundary Decision Report

This report is a form of process and rationale capture for the project. When decisions about domain boundaries are modified in the course of the project, the decisions and rationale are captured for future lessons learned, and as a way of documenting the change so that validation and modification of related workproducts can take place, assuring consistency of all ODM workproducts during modeling changes.

7.2.2.2.5 Sequencing

The Domain Interconnection Model is the driving workproduct for this process. The Qualified Exemplar Set documents the impact of the relational scoping of the Domain Interconnection Model on the Domain Exemplar Set. The steps can also be worked in reverse order, i.e., starting with *counter-exemplar* or boundary-case systems in the exemplar set and trying to find plausible domain categorizations for these systems. The Domain Boundary Decision Report will be an ongoing document; many of these decisions or renegotiations of domain boundaries may not be clarified until well into descriptive modeling.

7.2.2.2.6 Guidelines

The following paragraphs describe some of the primary types of domain interconnections recognizaed by the ODM method.

7.2.2.2.6.1 Operational Domains

Domains that would appear in a systems analysis context diagram, or a high-level system architecture, are related through interfaces in the operational environment of applications in the domain. For example, a text outlining program may exchange data with a file management sub-system and a window manager.

These interfaces often raise canonic design issues; for example, often there is a choice of either strong or weak coupling across an operational domain boundary such as a client-server relationship. Major system variants that appear in the domain model may be characterized in large part by design decisions at these boundaries.

Thus, the Operational Domains relation within the Domain Interconnection Model is related to a conventional system architecture, except that it includes the range of variability for these architectures. As a result, the Domain Interconnection Model will include a superset of Operational Domain relations, even where all of the domains modeled could not simultaneously interact with any single system in the domain.

7.2.2.2.6.2 Specialization Domains

Another important kind of relationship is one of specialization and generalization. These relations are generally orthogonal to the operational relations described above; unlike the latter, the boundary between domains related as specializations or generalizations does not appear in the operational environment of the typical target system.

For example, the text outlining domain supports features that are a subset of the hypertext domain. Thus an outliner can be considered a "special case" or restricted hypertext system. Similarly, a hierarchical directory browser uses a subset of outlining features.

We model these domain relations because specialization domains will usually support supersets of features supported by their parent domain.

This relation is echoed within the scope of the domain itself, and will be reflected in the layered domain architecture, which reflects different variants of the system or subsystem implementing the domain as a whole that can be obtained separately from the asset base.

7.2.2.2.6.3 Analogy Domains

Analogy domains are domains that may have influenced terminology, design choices, or documented explanations for system behavior provided to users for the domain of focus. A network of analogy relations between domains would thus more closely resemble a conceptual association network than a structural system components model.

For example, some text outliners derive terms and operations by analogy to family trees or genealogies. These terms show up in object and operation names within the domain, influence the explanation offered to users in training manuals, and may even have influenced some of the design decisions made by developers.

An analogy relation may reflect an earlier historical relationship between domains. Many automated systems retain the older terminology of superseded manual processes, even though restrictions implied by this terminology may no longer apply. So, for example, many computer typesetting systems retain terms and operations from hot-metal printing technology, although the terms imply a physical mechanism that is no longer used.

Probing for analogy domains in interactions with domain informants can become a kind of controlled brainstorming that pushes these analogies in surprising ways, and can help to unearth what may have been unconscious design choices and oversights. The technique is particularly useful for domain analysts who are **not** experts in the selected domain, acting in a consulting capacity with engineers and domain experts; it is one part of the DA process where lack of domain-specific knowledge can be a value rather than a limitation.

7.2.2.2.6.4 Other Domain Relations

Other domain relation types c[illegible]ed during this phase include domains related via the software engineering life cycle [illegible]ic to the organization and domain. Such relations are discovered by examining [illegible]n-specific software engineering processes as well as products.

For example, most application domains will have a related test data domain, although testing will typically be a function in the development and not the operational environment.

This approach reflects a view of DA as a kind of systems analysis of an application-specific software development enterprise [Sim90b]. It is integral to the goal of defining models supporting both constructive and generative implementations, since generative solutions often automate component tailoring and adaptation activities only observed by reflecting

on the application developer's process [Sim86]. The STARS CFRP offers an integrated view of this kind of "food chain" between domain developer, application developer and end-user environments [STARS92].

7.2.2.3 Determine Domain Coverage

7.2.2.3.1 Purpose

In Domain Definition and Domain Scoping we refine principles for bounding the domain that allow us to include or exclude exemplars. The Determine Domain Coverage process bounds the level of effort the project can devote to modeling this domain, given the modeling context, available resources, information sources, etc. Since domain models will be developed by analyzing and synthesizing domain information, choosing the types of information we will collect and the types of models we will build will have a significant effect on the final results of domain modeling.

7.2.2.3.2 Inputs

Inputs to the Determine Domain Coverage activity may include the following:

Domain Stakeholders Model

Domain Modeling Objectives Statement

Domain Definition Statement

Domain Interconnection Model

7.2.2.3.3 Resources

ODM Domain Information Sources Checklist

ODM Domain Model Types Checklist

ODM Modeling Formalisms Checklist

7.2.2.3.4 Workproducts

Domain Ontologies Model

Domain Modeling Formalisms Selection

Domain Information Types Model

Domain Representative Exemplar Set

7.2.2.3.5 Process

The following paragraphs describe the primary activities in this process.

7.2.2.3.5.1 Determine Domain Model Coverage

7.2.2.3.5.1.1 Select Model Types (Ontologies)

In this activity, modelers plan for the set of models to be built in the descriptive phase. For each anticipated model, the *ontology definition* helps to determine what kinds of individuals are the "subject of discussion" for that model. The subject of various models may range from concrete artifacts like particular code modules or other system artifacts, to more abstract conceptual entities such as types of errors, objects or operations in the domain. For each model, a "root category" name is proposed. This name can be thought of as the effective root for the taxonomy that will be built for that model type.

In formal modeling systems like the RLF, where the term "root node" has specific connotations, this terminology can be misleading. The Domain Ontologies Model can be thought of as almost a taxonomy unto itself, where the instances being modeled are individual descriptive models to be developed for that domain. Thus every instance in this model can be associated (whether or not with automated support) with another, separate model rooted at the same name.

Rationale. The need for a formal ontology definition step may be seen by reflecting on modeling experience of researchers in knowledge representation work. In semantic network- based systems, subtle differences in assumptions about the ontology of a given model can lead to great confusion. For example, one could define a Wines model that deals with wine regions (e.g., Bordeaux); another model might use the same term as the name of a generic category of wines, individuated by vintages. In this case, the latter model's ontology is of wine *vintages*. Still another model might describe individual bottles from a single vintage (e.g., for a quality control group), etc. In each case, though the general subject is Wine, the specific semantics of what an individual in the model stands for is different.

In the ODM method, where team-based modeling is a core discipline, lack of explicit shared concepts about the purpose and content of a given model can invalidate much of the modeling resulting in models of poorer quality, inconsistent sematics and lack of modularity.

Ontology Relations. The ontology definitions for the various models need to be defined in some relation to each other. While detailed connections between different models are not worked through at this point, the general relationships should be made clear.

Model Overlapping. When determining the various model ontologies to be developed for the domain, the ODM team may decide to deliberately model some data redundantly in multiple models. This could be for reasons of verification by independent development, to generate alternative views of the model entities for different audience/uses, or to assist in integration of the models at a later phase. The possibility of generating multiple views from a single model representation should also be considered.

7.2.2.3.5.1.2 Select Modeling Formalism(s)

The assumption throughout this document is that the modeling formalism chosen supports taxonomic semantics, with representation of specialization at a minimum. Certain specific models may be appropriately modeled in other formalisms. It can be helpful to

think in terms of a checklist of different types of formalisms useful for comparative modeling. Trade-offs to consider include the following:

- The closer the formalism used is to the culture of the application engineers within the domain, the easier it will be to obtain feedback on the content of the models.
- Some means for integrating the various models together semantically must be allowed for. One approach is to use a taxonomic modeling approach for all models to indicate variability, and use other specific modeling formalisms where appropriate to capture the nature of relations between entities within each model.
- Wide diversity in models will make integration more difficulty.

7.2.2.3.5.1.3 Adapt Starter Models (Utilize)

Purpose: 1) Acquaint team with the starter models relevant to selected model ontology and model formalism available in the reuse infrastructure, the environment, or as a requested asset from other projects. 2) Allow the team to be familiarized with knowledge specific to that ontology. For example, if a requirements model is to be developed, studying a generic or starter model for requirements is a good way to find out about the requirements phase itself: traceability, allocation, different categories of requirements, etc. 3) For starter models more particular to the domain (e.g., a feature model based on a domain-specific protocol or standard), this process insures that the ODM team will take such sources for models into consideration. 4) Whether or not previously developed starter models are available for particular model ontologies selected, the ODM team can create starter models to reflect the informal knowledge of the team before data collection begin.

7.2.2.3.5.2 Determine Domain Information Coverage

Full descriptive modeling of even a single system would be beyond the resources of most reengineering projects. In the case of domain engineering, where the scope of investigation includes multiple systems, with potentially wide diversity in software development methodologies, languages, etc., the modeling process depends on careful selection, both of a small set of representative domain exemplars (systems or subsystems spanning the full scope of the domain) from the Domain Exemplar Set, and a small cross-section of development workproducts from these representative systems.

7.2.2.3.5.2.1 Select Domain Information Types

In this activity, a cursory, high-level survey is performed of the kinds of domain information sources available for exemplars. This could include data from different life cycle workproducts, such as requirements, design, customer change requests; or different classes of domain informants to interview, such as requirements analysts, customer support personnel, field operators, etc.

This survey is not an exhaustive catalogue of specific workproducts available, but a kind of checklist of which types of workproducts or human informants are available for different exemplars. The question of availability for human informants may include issues such as the level of commitment of domain experts to time-critical programs, or organizational obstacles in obtaining access to certain stakeholders, such as customers outside the company. This information, along with the concurrent selection of specific model ontologies, is used

to guide a focus on particular types of sources from which domain information is to be gathered. This list will later be refined into the Descriptive Artifact Models.

7.2.2.3.5.2.2 Select Representative Exemplar Set

At this point in the process, the original Domain Definitional Set has been refined to include an Exemplar Set of systems/subsystems within the scope of the domain; this set has been further qualified during Domain Scoping so that borderline systems and even counter-exemplars are categorized into related domains. Now, a subset of the Qualified Domain Exemplar Set is selected for detailed study in the modeling phase. This subset is called the Representative Exemplar Set; the representatives can be thought of as data points within the variability space implicitly defined by the Domain Exemplar Set. The size and diversity of this set will depend on the resources available to the project and the stated objectives.

7.2.2.3.6 Sequencing

While this process is included as part of the Context-Settin sub-phase of Descriptive Modeling, many of the decisions made at this time can only be provisional. Data gathered during descriptive modeling activities will typically impact previous decisions made about particular artifacts to examine, informants to interview, and models to build. Modelers should beware of devoting too many resources to this process, given the provisional nature of these decisions.

7.2.2.3.7 Guidelines

There is a risk that modelers will assume domain informants can contribute only raw data which the modeling team will synthesize into models. Often, especially in mature domains, the experts will have their own ideas about taxonomic relationships within the domain information, and these "field models" should be considered carefully.

A related risk is making too early an entry into the formal modeling process, without the structure of methodical domain information-gathering to bound the process. The purpose of this phase is to gather model resources that can be used as inputs to the modeling process, not to commit to maintaining consistency with any of the models in particular. Extensive discussion of details of these models may be a sign that a shift from context-setting to modeling has occurred.

7.2.2.3.8 Issues

Some fuzziness of boundaries can be expected surrounding what distinguishes a "domain-specific starter model", as described above, from a domain information source. Similarly, which starter models are considered part of infrastructure, which part of the domain, can be a point of divergence. Such issues are mainly a question of making sure all sources are allocated to some category, and that the team agrees on a common interpretation.

Since Domain Model Ontologies include ontologies concerned with classes of domain workproducts, and the Domain Information Sources Catalog also concerns these workproduct types, it is reasonable to conclude that the choice of Workproduct Model Ontologies determines which workproducts will be used as data sources.

While to a certain extent this is true, the peculiar relation of domain modeling to the application development life cycle means that, in principle, workproducts from across the life cycle are available simultaneously to the domain modeler. Since each system will have its own gaps in coverage across these different life cycle products, domain modelers may need to do a certain amount of reverse engineering. In principle, workproducts from the various life cycle phases can be used in unconventional ways to derive information about other life cycle phases.

These workproducts can be thought of as a hologram, containing representations of domain entities and terms at different levels of abstraction. Our views of conceptual entities in the domain will be enriched by examining them from the viewpoints of as many stakeholders and life cycle phases as possible; for example, test suite scenarios may reveal details of system behavior never called out in the requirements. Therefore we may look at more information sources than those we intend to model as workproduct types in a domain model.

Note that this is different in principle from a view of a domain model as being essentially a model of requirements only. There are very good reasons for focusing on the requirements-level information in domain modeling. However, the ODM approach to descriptive modeling allows for the inclusion of domain information from any stage of the software life cycle.

7.2.3 Model Evolution (Enact)

This section describes the core modeling processes of the Descriptive Modeling phase: gathering and generating domain data, developing the domain models and validating the models with respect to the data.

7.2.3.1 Gather/Generate Domain Data

7.2.3.1.1 Purpose

We are now ready to begin the in-depth collection of information about the domain to guide development of the descriptive domain models. The purpose of this phase is to collect sufficient information about the domain to build the selected models in support of the overall ODM Project Objectives.

Most DA methods recognize the importance of gathering domain information from a variety of sources, including published documentation and textbooks, system artifacts, requirements and market studies, and interviews with domain experts and developers. ODM recognizes a richer and more comprehensive and multi-disciplinary set of potential information sources and information-gathering techniques, including ethnographic observation, team-based expert interviewing and review, and instrumented capture of rationale.

ODM also places greater emphasis on careful documentation of domain information sources, for several reasons:

1) to validate those models intended to be descriptive, i.e., based on precedents within domain exemplars;

2) to enhance the intrinsic value of the descriptive models as maps of domain expertise;

3) to provide traceability back to exemplar artifacts for the benefit of asset implementors.

7.2.3.1.2 Inputs

Inputs to the Gather/Generate Domain Data activity may include the following:

Domain Ontologies Model: defines what models the team expects to develop;

Domain Information Source Types: defines the types of artifacts the team expects to examine;

Domain Representative Set: defines the set of representative exemplar systems the team intends to examine in detail;

Domain Information sources: the artifacts and domain informants from which domain data will be extracted.

7.2.3.1.3 Workproducts

Domain Information Sources Catalog

Catalogs all sources of information consulted during descriptive modeling. When used in the planning process, can be integrated with priority-setting for different information sources, schedules of availability, and strategies for joint vs. individual meetings, etc.

Domain Dossier

Includes documentation of all information sources for the domain, including references to software artifacts investigated, journal articles or technical reports, and interview, meeting and conversation reports.

This document also serves as a focal point for tracking all the other workproducts and models generated for the domain. It is also a suitable repository for lessons learned about the domain information gathering process.

Prototype Domain Lexicons

Lexicons that document the informal understanding among domain-knowledgeable project participants. Diversity in interpretation are mapped only, not worked at this stage.

Precedented Domain Lexicons

Lexicons of terms in context, traced to a specific information source as mentioned in the above catalog.

7.2.3.1.4 Process

The process of generating data is by no means a mechanical one of reading through a document and parsing items of interest. Filtering the data sources and strategic selection of representative data are required at each step. The following paragraphs describe some specific activities within this process.

7.2.3.1.4.1 Identifying Domain Information Sources

Based on the survey of information source types available and the selected representative exemplars, a specific set of domain information sources to be investigated is proposed.

In ODM the task of domain information gathering is viewed in organizational terms. Domain knowledge is assumed to be distributed throughout the organization, not just within software engineering groups. However, few organizations are structured from a domain knowledge-centered perspective. While knowledge of some domains may be retained in well-defined groups within the organization (e.g., a central technology group responsible for acquiring and managing subsystem-level assets used in several applications), domain knowledge is more typically distributed across structural or functional boundaries within and across organizations.

For example, technical support personnel may contribute knowledge about typical ways in which end-users need to tailor the systems they receive, and this may in turn shape the defined boundaries of the domain model. End-users of systems are also potential sources of information about the domain.

The end-user environment will usually have been studied using conventional requirements analysis methods. In domain modeling we re-examine the end-user environment specifically from the standpoint of reuse. In particular, we search for recurring work processes or user-defined artifacts that could become the basis for reusable assets (e.g., user-defined formats, macros, style sheets, scripts, templates), or codifying user procedures for more direct support from the system itself. For domains with deployed systems as exemplars, we will also derive the benefit of users' interaction with existing technology as a source of data, as well as comparisons across divers user environments. Previous analyses will, of course, be valuable sources of domain information, to be tracked as artifacts.

Information Sources will span systems

Exemplar-specific workproducts will be drawn primarily from the Domain Representative Set. However, many important sources of domain information will provide information on more than one system, or information not easily correlated with one particular example system or subsystem. For example, expert developers with knowledge of more than one system will be particularly valuable informants as they will be more able to rapidly synthesize their experience into comparative statements about system capabilities. Survey articles will typically compare multiple systems; these sources will often contain established taxonomies of terms in the domain that can be used to create domain conceptual models directly.

In fact, the scope of many information sources will span not only multiple systems but multiple domains; hence some information may have begun to be gathered during Domain

Identification and Characterization. This also emphasizes the value in tracking domain information sources carefully, as they may be used in subsequent domain modeling projects.

7.2.3.1.4.2 Gathering Information Sources

This step will involve the actual research and acquisition process to get access to the various data sources for the modeling effort. It is important to make efficient use of time with domain experts. Artifact-based sources of information should be examined at least cursorily before extensive interviewing.

7.2.3.1.4.3 Artifact Analysis

Potential artifacts include the following:

1) formal system documentation (requirements documents, design, test plans, as well as code

2) benchmark results, test data, and other associated documentation (e.g., cross-references or metrics run on exemplar system workproducts);

3) informal documentation, such as checklists, glossaries, meeting notes, planning documents;

4) published sources, such as textbooks, survey and/or technical articles;

5) electronic mail discussions from user or developer groups;

6) technology forecasts, market surveys, customer-requests for enhancements

7) user manuals, training materials.

The Domain Information Sources Catalog is a survey of what information is available for various domain exemplar systems. The Domain Information Types workproduct provides a rough categorization of these artifacts, and a prioritization of which are considered most relevant of the specific objectives of the project. These two workproducts help to focus the domain information gathering task.

Not every artifact utilized as an information sources need be modeled in the artifact models of the Descriptive Domain Models set. These models describe commonality and variability in the structure and attributes of the domain artifacts themselves. This is most relevant when assets of the same or analogous type as the artifact will be developed for the asset base.

7.2.3.1.4.4 Ethnographic data collection

This step involves interviewing domain informants about the domain. The category of informant most familiar from other DA approaches and related disciplines from expert systems development are *domain experts*. By this is usually meant subject area experts with experience in implementing software systems in the domain.

Other categories of informants can also be interviewed, including end users from a representative sample of environments, requirements analysts and people involved with other phases of the software life cycle for the domain.

7.2.3.1.4.5 Filtering the data

Though some strategic decisions are made during the Determine Domain Model coverage process about what source documents and system artifacts to study, and what people to talk to, relevant information within these sources needs to be filtered out in order to be placed in the context of the evolving domain models. Information gathered from artifacts will need to be filtered considerably. For example, if reading through a user manual for the relevant portion of a system, a certain number of the operations described will be relevant to the domain, while others will be relevant to the overall system and not

Analogous to the choice of the exemplar set of systems, selection of data can be guided by two competing criteria: finding a data set representing the full range of diversity to be covered, or a data set that represents a typical sampling of the scope of applicability.

Where exhaustive information modeling is required, it will usually be advisable to consider automated support tools. For example, text scanning capabilities might locate all references to a particular term within a set of documents. In general, this level of traceability among system artifacts goes beyond what is feasible or desirable in the domain model proper.

7.2.3.1.5 Sequencing

Although this step is presented as part of the modeling cycle, the domain information gathering function really begins at the outset of the domain modeling project. For example, an empirical approach to doing Domain Identification and Characterization (within Domain Assessment) might involve interviewing some experts in the line of business, examining transcripts, and looking for lexical terms that suggest functional areas that domain experts themselves might consider domains.

For this reason, it is advisable to set up the infrastructure for tracking and managing domain information as early as possible. Data may later be migrated into various documents of the ODM workproduct set: for example, a term may be "promoted" from the domain lexicon to the business area lexicon if it appears to apply to more than one domain within the line of business and to have a generally accepted semantics within that line of business as a whole.

In general, though, domain data generation should occur after context-setting, so that the information is filtered appropriately and an infrastructure for capturing the information is set up at the outset.

What is not advisable is generating a large amount of data that must be then structured or modeled in a separate pass. The idea of tight coupling between the data collection and the modeling is a key feature of ODM.

The domain lexicon is one of the first workproducts produced during domain modeling, and in fact needs to be initiated at the start of the information-gathering phase. Since the lexicon includes linkage information between terms and information sources from which

they are obtained, the lexicon must be evolved in close correspondence to the Domain Information Sources Catalog.

New domain data can be generated after the descriptive modeling phase is completed as well.

7.2.3.1.6 Issues

In general, the approach in ODM is to identify interim deliverables as part of every process that can be utilized in their own right. In particular, domain information gathered for an ODM project may be considered valuable source material within the wider organization. It is wise to encourage interest in the materials, but with several provisos. First, the expanded context of interest should not affect the ability of modelers to perform focused, selective gathering of information. Second, where this specific project focus might skew the impact of the data as interpreted by other parts of the organization, this risk should be addressed with clear communication.

Rendering data more accessible to further study will usually, but not always, be seen as a positive contribution. In a few settings, the availability of previously undocumented information may become problematic. For confidentiality may be a factor with some domain informants (e.g., system users who do not want performance evaluations based on ethnographic data collected by the domain modelers). Any such commitments must be respected by the modeling team, if the project as a whole is to maintain credibility within the organization. This is one small aspect of a much larger set of ethical and organizational questions that in fact affect any researchers employing ethnographic data collection techniques [IRL???].

7.2.3.2 Develop Domain-Specific Lexicon

7.2.3.2.1 Purpose

The Domain-Specific Lexicon (DSL) is a collation of terms drawn from the domain, annotated to reflect information about each term and initial semantic relations such as synonyms, oppositional pairs and inverse operations (e.g., "promote" vs. "demote" headings in text outline processing). Most DA methods advocate gathering of domain terminology in some consolidated form. Perhaps the most thoroughly elaborated method in this respect is DAPM [???], which borrows lexical analysis techniques from the field of library and information science and includes a detailed process model for lexicon creation. The use of a lexicon can also be closely linked to the data dictionaries created as part of various structured analysis and object-oriented methodologies [???].

In the ODM method, the lexicon serves as an ongoing collection mechanism during domain information gathering and modeling, providing a snapshot at any time of the ODM team's emerging consensus on domain language and its meaning. Terms are drawn from domain information sources, including development artifacts (code, design, tests), documentation, literature, and informal terminology noted in interviews and meetings with domain experts. Terms can include phrases as well as single words; e.g., in the outlining domain "heading text" is a compound term used with the intended consistency of a single term. Relations may include synonyms, acronyms, and oppositional pairs.

7.2.3.2.2 Inputs

Inputs to the Lexicon Development activity may include the following:

7.2.3.2.2.1 Domain Information Sources

Information sources are *parsed* in order to generate data for the lexicon.

7.2.3.2.2.2 Domain Ontologies Model

Indicates the priorities of what models are to be developed from the lexicon data. This is not a hard filter on what is admitted to the lexicon, however; allowance is made for the domain data to suggest model types that were not initially planned for.

7.2.3.2.3 Process

7.2.3.2.3.1 Prototype Domain Lexicon

In practice, if the domain analysts are themselves knowledgeable in the domain, they may enter the analysis phase with a fairly large repertoire of existing terms as part of their informal knowledge. In this case, it is appropriate to first do a "prototype" lexicon based entirely on this informal knowledge, capturing the expected terminology for the domain without explicit references to sources. The Prototype Lexicon thus helps to capture some process history and design rationale for the team.

The Prototype Domain Lexicon can be created even before the Domain Information Sources Catalog is compiled. Later, the prototype terminology can be cross-referenced to tracked information sources.

If the prototype lexicon is established, two guidelines are important. First, these terms should be uniquely distinguished from later additions to the lexicon, even those later corroborated with referencing documents or interviews with experts. Studying the differentials, particularly, prototype terms eventually excluded from the Domain Lexicon, as well as key terms included that were not originally included in the prototype Domain Specific Lexicon, can in effect document the "surprising results" of the domain modeling effort (surprising, that is, for those knowledgeable about the domain). The same information will be valuable as process history information during the Reuse Learning phase to evolve the domain analysis process itself.

7.2.3.2.3.2 Precedented Domain Lexicon

Once the domain information sources have been identified and prioritized, and a prototype lexicon (if deemed applicable) has been established, the data gathering process can proceed according to a variety of detailed process sequences.

One can trace the process, following either a datum in a domain information source, such as a term used in an expert report, or following an existing term in the prototype lexicon.

Beginning with a datum in an information source, the first step involves identifying the datum as significant for the modeling effort. This involves applying the filtering criteria, generated through the choice of model types to be developed and the modeling goals (e.g., exhaustive versus selective modeling).

When a datum is identified, it is delimited, so that, for example, a requirements sentence in a document is separated from the surrounding context. It is next classified roughly with regard to which lexicon, if any, it belongs to. For example, most "terms" in the usual sense would be entered into a lexicon for the conceptual models for the domain.

Once the appropriate lexicon for the term is identified, the current version of the lexicon is inspected. If the term does not exist in the lexicon, it can be added to the lexicon along with a link to its precedent in the domain information source.

If the term already exists in the lexicon, and the usage appears reasonable consistent with the cited definition, then the decision as to whether to include the reference as an explicit informational link is made based on the existence of sufficient links already in the lexicon and the filtering criteria being applied.

If the term exists but has different interpretations in different references, this diversity of interpretation is noted by explicitly including the variant precedent in the lexicon. Insights about the commonality and variability in the usage of the term will have significance for the descriptive model. The eventual objective of the lexicon will be to normalize the term space so that a canonic term can be chosen and ambiguous terms resolved. The objective of the modeling effort will be to capture the variability in the domain in conceptual terms. Sufficient terms should be defined to capture all concepts requiring distinction within the domain.

7.2.3.2.3.3 Canonic Domain Lexicon

When the lexicon contains sufficient diversity for a given term, synonymous terms may be associated as a *term cluster* , for which a *canonic term* will be selected as a representative. The canonic term is used to normalize references to the term in other lexicon entries, and will later be used in descriptive models and model-specific languages, and even asset descriptions. Term clusters usually map synonym relations among the terms. However, other syntactic relations may be recorded in the lexicon. For example, acronyms, case, voice (e.g., active-passive) and plurality variants, etc., may be clustered into a single logical term with a designated canonic representative. Other relations might include antonyms (i.e., mutually exclusive terms) and oppositional pairs (e.g., converse operations). However, a fine line must be maintained here between syntactic relations and more interpretive semantic modeling. The lexicon is intended to serve as a syntactic filter, culling out purely syntactic kinds of relationships that would needlessly clutter later domain modeling stages. An analogy could be made to code generation systems that allow attributes to be specified on a semantic tree whereon all purely syntactic information (e.g., that needed for unambiguous parsing) is stripped away. (See Guidelines below.)

7.2.3.2.4 Sequence

An alternative approach to determining precedent for the lexicon is to begin with terms already existing in the prototype lexicon and to do a focused search for precedent usage of those terms within domain information sources outside the informal knowledge of the modeling team. This represents a specialization of the information-driven vs. ontology-driven process alternatives suggested for descriptive modeling as a whole; in this case, the choice is between data-driven and *lexicon-driven* identification of terms.

7.2.3.2.5 Guidelines

7.2.3.2.5.1 Limiting semantics in the lexicon

More sophisticated semantic relationships can be noted in the lexicon, with the following provisos:

- The lexicon structure is not intended for deep semantic modeling; it is too easy to inadvertently spill over into what is more properly handled in the modeling phase itself.
- Any semantic relationships recorded in the lexicon should be inferable from the information sources, along with the terms themselves,, rather than a second level of interpretation provided by the domain modelers. The lexicon should be confined to the domain informants' language and their perceived relationships between the terms. New relationships perceived by modelers should be noted informally and separately.
- The lexicon serves an ongoing integration and validation function, by providing a flat term space for the terms being created in the various models. If the lexicon is partitioned to reflect the divisions of the Domain Ontology Model or taxonomic relationships, some means should be maintained for checking the set of lexicon terms as a whole.
- Because the lexicon is developed before semantic modeling steps, those building the lexicon are less likely to filter domain terminology according to fixed types of models. For example, an analyst who knows she is looking for objects and operations might skip over terminology that is richly domain-specific yet involves relations, constraints, design issues, or other life-cycle workproducts. Careful tradeoffs must be made between this use of the lexicon as an inductive "completeness check" and the desire to filter data collection according to the models pre-selected for development.

7.2.3.2.5.2 Review by domain informants

The Domain Lexicon can serve as a transitional document for obtaining early feedback from domain informants about the modeling process. Since lexicons resemble glossaries and data dictionaries, concepts with which many technical workers are comfortable, it may be possible to create a formal and ongoing reviewer relationship with domain experts who might otherwise be difficult to draw into the modeling process in its full taxonomic sense.

7.2.3.3 Evolve Domain Model

7.2.3.3.1 Purpose

This step is where the central work of descriptive modeling takes place: creation and refinement of various domain models.

The purpose of the modeling, at this level of detail, is to create a model capable of accommodating all the variability in the domain information parsed from domain information sources in the representative exemplar set. This reflects the information-

driven approach to model evolution. Alternatively, the purpose may be to validate a given taxonomy with respect to a given information set. This represents the ontology-driven model evolution strategy. In either case, domain information items that are to be modeled will influence the formation of new categories, the reorganization of existing categories, or the restructuring of subsections of the model.

7.2.3.3.2 Inputs

Inputs to the Evolve Domain Model process include the following:

Domain Stakeholders Model: Used as a basis for the Feature Binding Times Model.

Domain Interconnection Model: Used as a basis for the Feature Binding Times Model.

Domain Ontologies Model This selection determines the models of focus for this task.

Domain Modeling Formalisms Selection: Determines the formal representations to be adopted for the various models.

Domain Dossier: Domain information sources, references, and information items. While some of this material will have been filtered through the Domain Lexicon, and, in fact, the lexicon is intended to serve as a transitioning document for domain information, the data sources themselves also need to be available as a cross-check.

Domain Lexicon: Ideally, the lexicon is available in canonic form before domain terms are incorporated directly into models. Otherwise, traceability should be maintained if possible.

7.2.3.3.3 Workproducts

Feature Binding Times Model

Descriptive Domain Models

Model Boundary Decision Report

7.2.3.3.4 Resources

ODM Guidelines for MSL Definition

7.2.3.3.5 Process

The modeling process as a whole can be performed in at least two different sequences. The first, the *ontology-driven* modeling approach, is focused around acquiring data to populate, or *inform*, a specific model. In this scenario, a modeler starts with a selected model, then sifts through domain information sources for objects relevant to this model. One advantage of this approach is that it is easier to control acquiring necessary and sufficient exemplar data for each model. One disadvantage is that the same information source may be analyzed numerous times, in the context of different models.

A second approach could be characterized as *information-driven* modeling. In this approach, the modeler begins with an information source, and attempts to distribute entities within that information source into the appropriate models. One advantage of this

approach is that certain information sources can be very thoroughly modeled, and may in fact yield useful model ontologies that were not originally planned for in the Select Domain Model Ontologies phase. One disadvantage of this approach is the difficulty in regulating the appropriate level of detail to model.

Hybrid strategies are possible. One possibility would involve a kind of "micro-matrix" team structure. Each modeler on the team is given ownership of a set of information sources (including, possibly, responsibility for interviewing particular people, i.e., domain informants, as well as code artifacts and documents); in addition, each team member is given ownership of a particular model or set of models. Interfaces between team members could be structured by the overall Domain Selected Ontologies Set (the catalog of the different models to be developed) and the Domain Information Sources Catalog (the planned scope for information gathering). Each team member would generate data, distribute exemplar artifact descriptions to other modelers, and receive exemplar data in turn for their own models. This approach requires a high degree of team coordination.

7.2.3.3.6 Sequencing

Any planned strategy will be provisional at best, due to the non-deterministic nature of data collection and modeling. ODM modeling activity consists of culling relevant domain information from various sources, determining the models for which the information is relevant, updating these models accordingly and validating the models produced with domain experts. Since information sources become available at indeterminate times (especially in the case of human experts), and each information source may contribute to multiple, closely inter-related models, ODM models will often evolve in parallel or iteratively, with modifications and reformations propagating through several models simultaneously.

7.2.3.4 Descriptive Domain Models

The Descriptive Domain Model actually consists of a number of separate models selected and tailored to suit the domain of focus, organization context and DA objectives. These models can be grouped roughly into three classes: artifact models, conceptual models, and contextual models.

In order to more fully understand the distinctions and relationships between these types of models, it will be helpful to first discuss a model which serves to structure their relations, the Feature Binding Times Model.

7.2.3.4.1 Feature Binding Times Model (FBTM)

The Feature Binding Times Model (FBTM) is refined from the Domain Stakeholders Model and relevant facets of the Domain Interconnection Model. It shows the producer-consumer chain that stretches from system developers to end-users within the domain of focus.

Purpose. The purpose of the FBTM is to document the varied contexts of both system development and system use where domain functionality or other differentiating features can be deployed. It is based on the notion of a *domain feature* as a differentiating behavior or characteristic associated with a system artifact that is significant to some domain stakeholder within a given context.

Rationale. We are accustomed to thinking in terms of a developer and an end-user environment for systems. However, these are merely two general cases for what, in each domain, will be a more specific set of contexts in which some aspects of the system functionality may be determined, or "bound". It may be helpful to produce a draft of the FBTM early in the descriptive process as a structuring framework for other descriptive models.

As an example, consider a code component. The executable produced from the component will function in the operational environment of the system, and various features can be associated with this context, such as system behavior under certain conditions and performance characteristics. Since these will presumably be of interest to the end-user (even if not directly visible), they will be valid features to model.

If the component's functionality was configurable at the time the system was installed, this would still affect the overall functional profile, but is effectively bound at a different point in the life cycle. (Admittedly, this stretches the normal usage of the term "life cycle", but "producer-consumer chain" is equally misleading.)

In descriptive modeling, we will want to know at what point in this spectrum a particular feature is bound. This may vary across systems, or even within a given system. The innovative modeling process will systematically explore novel binding times for existing features by methodically *shifting* featrues associated with one context across *contextual interfaces* to new contexts (e.g., compile-time to run-time). During prescriptive modeling, various examples and potential innovations will be evaluated for usability and feasibility, and a subset of these selected as specifications for implementing assets.

To begin this process, we need a model of the various contexts themselves and their relations. Note that this model does not contain references to specific features, only to a set of contexts within which features will have different degrees of visibility and malleability for various stakeholders with various interests. Features will be linked to specifc contexts when descriptive feature models are developed, during Domain Model Integration.

The reason this is an explicit modeling step, rather than just a generic model, is that the set of producer-consumer contexts will vary for each domain. For example, the extent to which software functionality is embedded in hardware components will affect where configurability options are available.

The FBTM refines the Domain Stakeholders Model, and information from various relations in the Domain Interconnection Model, in part by addressing a lower level of context granularity. For example, user interface-oriented software will typically have a number of distinguished binding times even within the operational environment of the system, e.g., system install time, session start-up time, tailorability through "preferences" menus, dynamic defaults, explicit commands, down to context notions as specific as "next-operation-only overrides". Embedded software or firmware in a printer or other peripheral product, by contrast, may require specialized personnel such as field technicians or site system managers, each of which may have particular features of interest and capabilities to shape the performance of the system.

7.2.3.4.2 Model Boundary Decision Report

Clearly, at low levels of granularity, it may be difficult to distinguish a feature binding time or feature binding context from a specific operation or object. This could lead to confusion, particularly if different sub-groups have been allocated responsibility for different models. Negotiating clear boundaries between the scope of different models will be an ongoing challenge in domain modeling; suggestions for partitioning models made as part of the ODM method will need to be interpreted in the context of each project. As such boundary issues are encountered and resolved, it will be good practice to document the process and rationale behind the resolution in a Model Boundary Decision Report, analogous to the Domain Boundary Decision Report used during Domain Scoping. This form of report will be useful for documenting boundary issues between other models as well.

7.2.3.4.3 Artifact Models

Artifact models describe the commonality and variability among physical elements of the software systems within the domain, including code components or architectural features of existing systems, as well as the attributes and structure of other development artifacts such as requirements, design and test documents. In simplistic terms, these models describe entities that system developers use as inputs or create as workproducts, as opposed to entities representing objects and operations in the run-time environments in which domain systems perform.

One rationale for modeling the commonality and variability among exemplar artifacts is that they represent prototype versions of what could eventually become assets in the domain asset base. Modelers need to understand the range of variability in these artifacts, to understand how they can be re-engineered. In cases where no re-engineering is planned, the artifact models may still provide useful pathways back into the systems that have been studied.

The FBTM concept helps to clarify the intent behind the artifact models within the descriptive phase. Descriptive artifact models capture the commonality and variability in various workproducts from across the software life cycle for domain exemplar systems. Focusing, in turn, on every feature binding context in the FBTM, we ask the questions: What workproducts are being used and created in this context? Who are the stakeholders that are creating and/or using them? What characteristics of these artifacts are significant to these stakeholders? If such workproducts were re-engineered as assets, what differentiating characteristics would potential utilizers want to know about?

The models are called "artifact" rather than "workproduct" models to emphasize the fact that more than just formal workproducts may be investigated, that these information sources will be used by the domain modelers for different purposes than those for which they were originally engineered, and that therefore such artifacts may be decomposed into smaller significant units than would be relevant for application engineering.

Example: Consider the requirements definition phase for a domain within a large-scale software development environment. Requirements statements discuss obligatory system features, performance criteria, contractual obligations, etc., all of which is part of the conceptual model for the domain. But the various formal and informal documents used in the requirements process can be examined in their own right. These artifacts might include

sets of specific requirements lifted from previous system requirements (literal "requirements reuse", proposal boilerplate, analysts' informal checklists of essential requirements, "gotchas" (typically missing requirements), results of negotiations on requirements, etc. Each of these types of artifacts is a potential source of domain information that will propagate into other models. Each is a potential type of asset that might be included in the asset base. Each, therefore, will be associated with a set of significant features.

For example, by studying different requirement checklist exemplars, a standard checklist for the domain could be created as an asset. Features for such an asset might include whether the checklist is written for an expert or novice user, whether its criteria can be scanned for with automated tools such as requirements parsers, etc.

7.2.3.4.4 Conceptual Models

The second broad category of models developed during the descriptive phase are termed conceptual *models*. Whereas artifact models describe concrete system development artifacts within some feature binding time context, conceptual models describe abstract concepts or entities, primarily within the operational environment for the systems within the domain. Artifacts (or assets) can be associated with particular features; conceptual models provide the groundwork for describing the intended semantics of those features.

In some senses, a software system is (or embodies) a model, a representation of agents, objects and processes in the real world sufficient for its purposes. In developing conceptual models for the purpose of software reuse, we are *not trying to model this real world directly;* rather, we are modeling the filtered representation of that world, as shared by developers and users of software systems in the domain of focus.

It is important to distinguish this modeling approach from that of traditional artificial intelligence or knowledge-based systems development. In some of these applications, particularly those in which the knowledge-based technology is intended to be deployed in the operational environment, there is at least an implicit attempt to "model the world". The aims of domain modeling are more modest, though still knowledge-based in a sense: to model the knowledge of expert software developers (as well as users and other stakeholders) about software systems in a particular application area.

The following paragraphs offer some suggestions for possible conceptual entities to be modeled. These are intended to be neither exhaustive, nor prescriptive; it is assumed that each project will tailor this list to suit its own needs, and will discover other conceptual facets appropriate for the domain of focus. It should also be noted that different exemplar systems may represent certain conceptual entities in diverse ways that suggest allocation to different models. To some extent, partitioning into objects vs. operations, for example, already embeds significant design decisions. Like all descriptive modeling, the approach here is to capture the range of such representations for the exemplars modeled, leaving the consideration of alternative modeling strategies to the innovative and prescriptive phases.

7.2.3.4.4.1 Objects

The object taxonomy for the domain focuses on entities that are managed by systems in the domain. Some criteria to consider in evaluating "objecthood" are: 1) the possibility of creating multiple instances of the entity as part of the function of the domain; 2) the

possibility of distinguishing co[illegible]f the entity, even when all attributes are equivalent; 3) the ability to separately mani[illegible] sub-components of the entity, and for the entity to participate in larger ensembles as a sub-component. These are general guidelines and are not intended to be exhaustive.

In modeling objects, both taxonomic and structural (or, for larger-grained objects, architectural) relations can be modeled. Taxonomic relationships can be defined between terms for objects when the terms represent different degrees of knowledge about the objects or when the specialization applies to a subset of the objects categorized by the parent. For example, in the outlining domain, different systems use terms to describe any outline heading, in contrast to those that have sub-headings beneath them in a given outline. An issue to track closely here is possible overlap between the notions of object and state, since one modeler might term an object in a different state as an object subclass.

Object-oriented modeling techniques may prove useful here, with the following caveats. First, ODM does not assume nor require use of object oriented techniques. Second, object-oriented assumptions such as the encapsulation of methods with data should not be imposed upon the domain model if the exemplars studied do not exhibit this organization. Whether this would always improve the designs within any domain is a subject for debate beyond the scope of this report; but in any case, such restructuring would defeat the rationale behind descriptive modeling of the domain. Note: as mentioned in Section 6.3.1, (the ODM Principles Section on domain vs. system modeling), restructuring of artifacts using various system modeling techniques is a legitimate strategy to facilitate comparison of analogous artifacts and concepts. These restructured artifacts should be distinguished, both from **original** exemplar artifacts, and from the domain models proper.

Third, specification of taxonomic relations (i.e., specialization with inheritance) on object categories has slightly different meaning in the domain modeling context than might be assumed by those with an object-oriented background. The driving motivation behind domain modeling is to produce asset specifications with specified semantic relationships, so that assertions made about parent asset categories can be confidently asserted about specializations as well. Therefore, specialization among objects (as well as for other conceptual entities) has *definitional import* and not operational import (e.g., code-sharing, etc.).

7.2.3.4.4.2 Operations

Operations, functions, or behavior concepts form a complementary model to the objects model. These concepts map the most directly to the intuitive notion of features as user-visible functional capabilities. Terms for the operations are placed into a taxonomic model, following similar criteria to the object taxonomy.

Examples of operations from the outlining domain include terms such as "promote" and "demote", "expand" and "collapse", "create new heading" etc. These terms are modeled in a taxonomy that includes specialization relations, e.g., "collapse heading" vs. "collapse family" which acts recursively on all subtrees. The model also includes attributes or relationships, e.g., operations have an associated role for "scope of effect", which ranges over another sub-taxonomy of possible ranges for operations, such as "character", "word", "current heading text", "whole outline", or "selected region" (probably elements in the object model). Aggregation relations can also be modeled to represent operations that are

logically equivalent to sequences of other operations. For example, "create aunt" could be modeled as a composite of "move to parent", "create sibling".

In these examples, it will be noted that object terms are freely intermixed with operation terms. In general, starting from pure description of domain information sources, operations will not be fully factored out in the form eventually desired in the domain models. The descriptive approach requires that we capture the operations in the form they exist within the exemplar systems (even thought this may appear implementation-dependent in the extreme) and to introduce categorizing concepts integrating the exemplars in the taxonomy. In this way we remain explicit at all times of where in the modeling process our own terminology and interpretation has entered in.

The operations or functions model is thus distinct from a functional model for a specific system, such as a data-flow diagram or a state transition diagram. These types of models are considered artifacts in ODM terms, because they describe the behavior of a single system, rather than commonality and variability **across** systems and because they are assumed to be available as domain information sources. The conceptual entities in models for objects, operations, relations, etc. will most likely map to individual components of these other models; thus the conceptual models form a kind of bottom-up analysis of domain terms from these sources, and facilitates their comparison as well.

Object-oriented aficionados will note the separation of object and operation taxonomies, in seeming violation of object-oriented encapsulation principles. This arrangement neither conflicts with nor supports object orientation in the eventual implementation of assets; concepts in these taxonomies will later be integrated in domain feature models. Separate *modeling of the objects and* operations taxonomy facilitates the later exploration of orthogonalized combinations of objects and operations in the innovative modeling stage.

7.2.3.4.4.3 Relations

The objects and operations taxonomies should yield a set of terms that can be used to specify data structures, environmental configurations or other types of scenarios for the domain. These structures induce further terms for specific kinds of relations in the domain.

Example: In the outlining domain, the term "headings" denotes a basic object in the outline structure; the terms "sister" or "sibling" are synonymous terms denoting a particular relation between headings. Relations are, of course, linked to the objects and/or operations to which they apply in exemplar systems. In addition, relations can have their own taxonomic relationships; for example, the relation "subsequent sister" specializes "sister", and the (interpreted) term "immediately subsequent sister" would specialize the first term, applying to only the first of possibly a list of headings following the current heading.

7.2.3.4.4.4 Constraints

Constraint modeling can be more or less integrated with the other models, depending on the degree of sophistication of the modeling representations selected and the specific constraints they support. Constraints are important to document for several reasons. First, they can simplify other aspects of the model, by allowing the taxonomic model to generate a superset of the instances intended for coverage, then "carving away" the excess through the use of constraints. Thus use of constraints, combined with features such as multiple

inheritance, can help to control the potential combinatorial explosion that always lurks as a danger in domain modeling, where numerous potential configurations are to be considered.

Constraints are also valuable in documenting what is "not in the model for a reason", so to speak. When what might be considered a feature in one context is specifically required to be absent in another context, it can be hard to distinguish accidental from intentional absence of the capability. These can lead to quite complex modeling issues, particularly in terms of the semantics of inheritance.

Example: In the outlining domain, one exemplar enforces constraints that disallow the creation of outlines in "improper style" (i.e., a heading of level 3 immediately following a heading of level 1). Other programs, patterned more after the notion of styles in a document format, allow such configurations. If the intended use of the tool is as a strict outline manager or idea processor, the more restrictive version of the tool may in fact be more useful.

7.2.3.4.4.5 Conditions/States

With the definition of objects, operations, relations, and constraints, a basic vocabulary is created that allows the description of conditions or states. This model is closely connected to the error semantics model and environmental characteristics model, described below. Each condition or state description may integrate concepts from the other models. The condition statement may be describing system conditions or conditions in the surrounding environment for the system.

This model should be distinguished from a more familiar use of states within state transition diagrams. Here, the intent is not to specify the behavior of a single system under a set of input conditions, but to classify the various significant condition and state descriptions occurring within the exemplars. The model could therefore theoretically include sets of conditions that could not all be realized by a single system. Specialization on conditions implies, again, an increase in the information available about the state.

7.2.3.4.4.6 Error Semantics Model

Error values often form an implicit taxonomy or classification scheme for unusual system conditions. Typically, system designers model such error conditions in a relatively flat error coding system with informal semantics. Some programming languages like Ada offer more semantic control over errors through exception-handling mechanisms, etc. Even in older languages, however, some taxonomic relations between error conditions can be deduced from error message text, conditional statements, error processing and user documentation. Of particular interest are errors that can be considered "special cases" of other error conditions, and understanding which errors take precedence when multiple error conditions occur simultaneously. These distinctions can lead to subtle variants in the behavior of components when lifted out of their original context of use.

The FBTM bears a close relationship to the error semantics model. Some errors can be generated at different points in the operation of the system; e.g., at compile-time versus run-time. In understanding the variant profiles of a particular functional capability, it is important to understand this variability. This will become particularly important when generation techniques are introduced, since these techniques have the advantage of making

earlier error correction possible. Thus, differences in error-handling policy can be a significant differentiator among system components.

In this regard, one possible fallacy regarding reusable component engineering must be addressed. It is sometimes claimed that components designed for reuse must be built to be extremely robust. In fact, robust components of this kind may be unusable in certain circumstances. Design for reuse entails consideration of variant implementations ranging from "weakly" to "strongly" error-resistant components. A weak component would check for very few error conditions at run time, and would make correspondingly strong assumptions about the execution context. Strong components, by contrast, would make fewer assumptions and test more aggressively. Such robust components might be excellent for prototyping applications, educational environments, and programs not too driven by performance constraints. Their weaker cousins, however, could be used in environments where the penalty for unnecessary run-time error-checking is exceedingly high.

7.2.3.4.4.7 Environment Characteristics Model

Most system engineering models emphasize modeling the operational system itself. Assumptions about the environment in which a system will operate are sometimes only informally documented. In engineering for reuse, where multiple contexts of operation are a given, the potential variability across those environments must be considered carefully. Unlike states or conditions, which are more transitory and are used to specify the behavioral semantics of individual features, environmental characteristics are more likely to affect the configuration of domain assets at a global level.

One key objective in this modeling activity is to identify aspects of requirements or system descriptions that encode information about the anticipated characteristics of the world outside the system. An example would be timing "requirements" expressed as a minimum or maximum interval to be handled between two external events driving the system. In this instance, the requirement is indirectly stating a set of expectations about the environment; these expectations can be modeled and compared with expectations embedded in the design of other exemplars in the domain. As with the error semantics model, an implementation that makes fewer assumptions about environmental characteristics in not inherently more reusable.

7.2.3.4.4.8 Stakeholder Profile Model

The last conceptual model described here is closely related, both to the Domain Stakeholders Model and the environmental characteristics model within the conceptual model set. For certain sets of operations, it may be necessary to model different classes of users for the system (or different classes of developers, etc.) Whereas the Domain Stakeholder Model distinguished stakeholders and their relations, in broad, CFRP-based terms, this model might involve taxonomic modeling for each class of stakeholder identified in the other models. This model may be used when classifying features during Model Integration and Domain Domain Model Interpretation, identifying, for example, sets of operations granted to privileged vs. guest users, expert vs. naive users, or users carrying out certain distinct kinds of real-world tasks requiring different levels of system support.

7.2.3.4.4.9 Overlap between Artifact and Conceptual Modeling

There will be potential areas of overlap between artifact and conceptual models. One such borderline area has to do with artifacts in the end-user environment.

For example, in the outlining domain, the concept of an "outline" is clearly fundamental to the object model, since outlining programs manipulate data structures that represent outlines. This would imply that the term "outline" belongs in the conceptual objects model.

However, an outline can also be considered a document, and as such is a potential artifact and even asset. Consider, for example, value-added resellers who work with given outlining programs, produce specialized outlines in particular subject areas and market these outlines themselves as products. Such stakeholders will clearly be interested in outlines as objects to be supported by the outlining tool; on the other hand, they will also be interested in outlines as artifacts or assets within an asset base.

A moment's reflection will reveal that this seeming conflict is actually an appropriate representation of the state of affairs in this scenario. An outline must be considered as both a domain object and an artifact, and each view will impose unique considerations on the domain model.

In general, this issue will emerge whenever systems within the domain of focus, as part of their functionality, manipulate software artifacts such as structured files of data and provide access to such artifacts to users.

7.2.3.4.5 Contextual Models

The third category of model addressed in descriptive modeling are termed *contextual models*. Contextual models attempt to capture information about processes, rationale, decision histories, trade-offs, and issues surrounding elements in the other models. Usually, rationale will be associated with variants in artifacts within the domain; but in principle, contextual information could also be recorded about conceptual entities such as objects and operations (e.g., when might a particular operation be useful, what are its performance penalties). This is the basis for proposing the three modeling classes as relatively independent, yet interdependent.

The artifact model, refining as it does the Domain Information Sources Catalog, are the easiest to attempt to prescribe in advance. Conceptual models will emerge to some extent out of the domain information interpretation; however, some general projections can be made based on the intended scope and objectives for the project. Contextual models, however, almost by definition, need to be defined opportunistically and on the basis of discoveries during information gathering.

The extent to which contextual information is available to be modeled will depend to a large extent on how much direct contact the project team can have with domain informants. In working from typical static sources (e.g., documents, code, etc.) it will be difficult to recreate contextual information unless it was documented as part of the creation of the workproduct. (The ODM method itself is designed to maximize the contextual information that is preserved along with the direct results of domain modeling.)

7.2.3.4.6 Model-Specific Languages

Every model that is developed during Descriptive Modeling will have a strong taxonomic element, in that the purpose of each model is to describe significant commonality and variability within the representative exemplar set.

Direct representations of these taxonomic relations are critical for this modeling process. However, there can also be a need for alternative representations, particularly in a more formal, textual notation.

This approach is in keeping with the notion that generative modeling is a core discipline in ODM. Since every model evolved during Descriptive Modeling will eventually become or be transformed into a *model asset*, it is appropriate to consider alternative representations for the models and for instances of categories in those models.

A Model-Specific Language (MSL) serves several purposes. First, the exercise of defining the MSL serves as a cross-check to the modelers that certain semantic constraints have been clearly defined. Taxonomic modeling alone may allow some of these constraints to go unrecognized. For example, in defining a formal notation, or MSL, for the Domain Interconnection Model in ODM, the notation definition raised the issue of whether two domains could be related with two or more distinct types of relations.

Second, the MSL provides a convenient notation for encapsulating descriptions of particular instances for various models. For example, if an object taxonomy has been developed for the domain, and there is a desire to be able to specify example objects (somewhat analogous to test data), the MSL can provide a concise syntax for such descriptions in a form that can be managed as a document in its own right.

Third, by being defined in at least a semi-formal syntax, the MSL allows for ease in validation, comparison, cataloguing and configuration management of these descriptions. By observing certain conventions that allow commenting, importing/exporting , etc., the specifications can become reusable components in their own right. (The formal notations for ODM workproducts included as ODM resources are themselves examples of MSLs, defined for the admittedly unstable domain of domain modeling methods.)

Definition of MSLs does **not** imply a commitment to implement generative tools to support the MSL, although this was often be a logical follow-on activity. The MSL can serve a very useful purpose merely as a more formal documentation approach. However, defining the syntax formally enough to serve as input to a generation system provides a useful standard to ensure consistency across multiple languages, and will help to highlight semantic issues such as forward references and partial specifications.

7.2.3.5 Model Domain Information

7.2.3.5.1 Purpose

Once domain information has been gathered and encapsulated, and the relevant models have been developed to reflect significant variability in the data, selected data items are allocated as instances within the models. This allocation is to provide traceability from the descriptive model back to the source artifacts.

One reason to preserve this traceability information is to validate the models. By ensuring that every category and every specialization has an exemplar, we know that we have added only categories that are descriptively justified.

Another reason for traceability is to support the asset implementation effort at the end of the ODM modeling process. Asset implementors may want to examine or even reuse (i.e., leverage) artifacts from exemplar systems in reengineering assets for the domain. Characterizing the exemplar data in terms of the descriptive models provides the necessary linkage to support this usage of the artifacts. (The model assumes that substantial restructuring of artifacts would be required to turn them into managed reusable assets.)

7.2.3.5.2 Inputs

Inputs to the Model Domain Information process may include the following:

Domain Dossier: contains the information sources (artifacts and transcripts of interviews with domain informants) from which domain data items are extracted.

Descriptive Domain Models: not yet necessarily populated with links to exemplar artifacts.

Domain Lexicon: establishes linkage from terms used in descriptive models, through lexicon entries, back to precedent in exemplar artifacts.

7.2.3.5.3 Workproducts

Descriptive Domain models, populated with linkage information to exemplar artifacts.

7.2.3.5.4 Process

When we model exemplar artifacts, we first determine what model or models the artifact belongs in, by comparing its artifact type to the model types established in the Determine Domain Model Types activity. For each model type addressing the type of the artifact within its ontology, we classify the artifact according to the top-level taxonomy for that model. A code component might be classified according to some general criteria (language, size of module, other aspects of concern to the developer) and according to a domain-specific taxonomy of the objects and operations implemented by the component. In a similar way, a requirements component (i.e., the actual textual statement that constitutes a customer requirement) might be first classified as a functional vs. a performance requirement, an operational vs. a project requirements, and so on; then further partitioned in terms of domain-specific categories (particularly functional requirements).

Once the artifact has been classified to the most specific category possible within the model, a decision is made as to whether the artifact should be included as an instance in the model. This decision will depend on whether the objective of the model was full or representative coverage of the data.

7.2.3.5.5 Sequencing

Populating or *informing* a model involves adding object descriptions into model at the appropriate categories. This is distinct from adding a new specialization or a new relationship to the model itself. Often addition of an object will necessitate an adjustment

to the model structure itself; this can be considered part of the evolution of the model. Clearly, then, evolving the domain model and modeling domain data are closely intertwined operations.

7.2.3.5.6 Guidelines

Modeling Informal Features:

The ODM method takes a relatively formal approach to domain features, deferring them to the Model Refinement sub-phase of Descriptive Modeling. However, functional features may be the most familiar way for experts to talk about the domain, expressing their knowledge in their own terms. Therefore, modelers should be alert for artifacts or ethnographic data that suggest *informal features* as recognized by domain stakeholders. This information should be captured and preserved in some way, so that these informal feature concepts can be cross-checked against the more formal features to be developed later.

Criteria for Completeness:

Modeling is, by its nature, an activity that is difficult to bound on the basis of content. Modelers should remember that a primary rationale for the definition of the Domain Representative Set is to provide empirical exit criteria for the modeling process. The models will be inherently incomplete, yet, by documenting the Domain Representative Set from which the exemplars are drawn, the limits of the model are also well-documented and easily extended at a later time.

7.2.4 Model Refinement (Learn)

This section describes the third major sub-phase of ODM descriptive modeling, Model Refinement. This phase is analogous to the Learn process family of the CFRP Reuse Management idiom, as specialized in the context of descriptive modeling. Consistent with this learning role, processes within this sub-phase deal with validating and integrating the descriptive models developed, exploring innovations arising from the modeling work, and consolidating these innovative possibilities into recommendations to be passed on to the Prescriptive Modeling phase.

The three processes within this sub-phase are termed Model Integration, Model Interpretation and Model Innovation. During Model Integration, separately developed models are unified and checked for cross-consistency. One primary means of integrating the various descriptive models is through the Domain Feature Models, formalizations of informally described features gleaned from exemplar artifacts such as requirements documents and interviews with domain informants. During Model Interpretation, the context and rationale is captured or developed for the clusters of feature patterns observed in exemplar systems. In a sense, this is the process wherein modelers build valid theories about why systems in the domain are structured as they are. During Model Innovation, the modelers for the first time are free to go beyond those feature combinations that can be empirically observed in existing systems or definitive requirements for new systems. Model Innovation allows new possibilities for domain functionality to be revealed. The primary output of the entire Domain Refinement sub-phase is therefore a set of features and feature variants (at both the functional and the architectural level) based on but not confined to purely descriptive features. The following subsections describe these processes in more detail.

7.2.4.1 Model Integration

7.2.4.1.1 Purpose

The first purpose of the model integration phase is to validate that the separate modeling efforts undertaken during descriptive modeling have produced consistent and complete results. Since the modeling task is partitioned into separate models that may be developed by independent teams, either in sequence or in parallel, it is important to have a closure step that verifies the integrity of the models. This is in addition to the validation done on individual models, and can be compared to unit vs. integration testing in conventional software development.

A second, and closely related, purpose of the model integration phase is creation of the Descriptive Domain Feature Model(s). These models reduce the detailed information in the Descriptive Domain Models to a set of distinguishing features that represent variability in the domain deemed significant by various stakeholders. Features are the central abstraction mechanism in ODM by which the functionality in artifacts is restructured and allocated to assets.

7.2.4.1.2 Inputs

Inputs to the Model Integration process may include the following:

Descriptive domain models resulting from the modeling phase.

The Domain Ontologies Model produced in the Determine Domain Model Coverage process, in particular, the ontology for the Descriptive Feature Model, which includes the linkage between the conceptual and contextual models.

7.2.4.1.3 Workproducts

Integration-Validated Descriptive Models: Models resulting from the previous description phase, validated and adjusted if necessary to satisfy integration requirements. In particular, any redundant or overlapping representations have been reconciled.

Descriptive Feature Models: Model integration can proceed on a feature-by-feature basis. The output of this process may therefore be a set of independent feature taxonomies. Ideally, the structure of a feature should link all of the models defined for the Conceptual Model layer in the Domain Model Ontologies.

Learning Recommendations : Lessons learned in this phase might include methods to better enforce consistency across independently developed domains. These methods could include infrastructure capabilities such as increased automated support, better training in modeling, or sounder procedures for check pointing parallel work.

7.2.4.1.4 Process

Individual feature specifications are linked to related variants within descriptive feature models. Each of these models provides a somewhat self-contained taxonomy of variants for a particular feature of interest. Certain high-level features may become part of the domain definition, other more complex feature models may spawn subdomains, effectively

deferring the detailed modeling of variation for that feature. For example, file system security may be a feature of many applications, yet prove both complex enough and reusable enough in its own right to warrant separate status as a subdomain.

7.2.4.1.5 Sequencing

Validation activities may occur on a model-by-model basis, and may be incremental with each cycle of gathering data, refining the model structure, and modeling the data within the structure. As a precursor to other integration activities, however, a more global and thorough validation cycle is assumed, that leaves the descriptive models as a set in a relatively stable and consistent state.

Before beginning the transformation to a domain architecture, it is advisable to validate the domain model in its descriptive form. One advantage of providing this checkpoint in the domain modeling process is that the descriptive domain model can be used for training and other purposes, independent of later modeling stages. The model should therefore be validated and stabilized in its descriptive form, whether or not the architecture phase is to be undertaken. It may also be worthwhile to maintain the descriptive model to be consistent with the architecture, if changes in the architecture result in new insights that would affect the original domain description.

7.2.4.1.6 Guidelines

Integration

When reviewing the models for the purposes of integration, there are several factors that should be considered:

- **Consistency of model information**: Similar data should not be modeled in dissimilar ways in different models of the domain. Wherever possible, uniform conventions in naming, model structure, etc., should be followed by the team. Inconsistencies should be noted, recommendations for future strategies developed (see the Learning Recommendations workproduct), and the inconsistent model content altered via negotiation among the respective modeling teams with ownership of the models in question.

- **Unplanned redundancy**: The integration phase should insure that any redundancy discovered between models in the domain was planned redundancy (preferably in the Determine Domain Model Coverage phase), and that the rationale for the redundancy has been addressed in the integration plan (e.g., multiple views validated against different audiences). Unplanned redundancy should be addressed by consolidating and deleting all but one of the redundant sections, arranging appropriate interfaces to the sanctioned model variant from the other models.

- **Gaps in coverage**: The models should, together, cover the intended aspects of the domain at a consistent level of detail. Some gaps will be a consequence of decisions made in the Determine Domain Model Coverage phase; other gaps may indicate the need to revisit some of the models with an expanded set of input data.

Feature Model Evolution

A *feature* can be characterized as an operation on some domain state (object configuration, relations, environmental conditions, etc.). In this sense, a feature can be thought of as a restatement of a testable requirement in the form of a capability statement that could be associated with a given software artifact.

Each significant feature may generate a sub-model of *feature variants* that can be treated as a semi-independent Domain Feature Model. Features need not share identical profiles with respect to their linkage to other models. Each model describes the range of variability for the feature of interest within the Domain Exemplar Set. This implies that features described in the Domain Model Refinement phase are **still** descriptive, in that each feature variant must have some precedence in an exemplar.

Although each term in the various conceptual models will be traceable back to some precedent in an exemplar artifact, this traceability will not necessarily propagate directly to the associated feature. That is, the feature in question may not be supported by the exemplar system that served as precedent for the constituent conceptual terms that describe the feature. Conversely, some features may be derived directly from domain information sources without being first synthesized from the underlying conceptual models. Nevertheless, features are not merely invented at this stage; they must correspond to some capability within the exemplar set. However, a feature may not always have precedence in an implemented system; it could be linked to a known requirement for a system still under development at the time of the domain modeling project.

Features can be roughly categorized into three classes: *defining features, differentiating features,* and *anomalous features.* Defining features are features that, at a given level of abstraction, are common to all exemplars for the domain. Differentiating features are features that have variants associated with different exemplar artifacts—usually from different exemplar systems. Anomalous features are those exhibited by a single exemplar.

For example, in the outlining domain, if all exemplars exhibit the capability of collapsing sub-headings, then this feature, **at this level of description**, could be considered a defining feature for the domain. On the other hand, if some programs did not have this capability it would become a feature of comparatively greater interest.

Defining features, when modeled, naturally are promoted to being attributes of the top-level category within their respective domain model. Unless they are required for the specification of specialized variants, they can be left out of the model and included in the features description of the Domain Definition Statement.

Similarly, anomalous or unique features can either be represented as a trivial variant in the model or excluded from the model and included with the description of the exemplar. For example, in the outlining domain, if only one exemplar system had the capability to collapse sub-headings, a feature model or sub-model focused on this feature would have little empirical basis for determining at what level of detail to model this feature.

Anomalous feature information should not be ignored, however, on the presumption that there is insufficient commonality to warrant their eventual inclusion in assets. It is premature at this stage to prioritize features for reusability on the basis of their relative distribution within the exemplar set.

Note the distinction between models that describe only commonality, models that describe commonality and variability, and models that would be required to model anomalous variability as well. In general, the feature models built in the Model Integration phase constitute differentiated features.

7.2.4.1.7 Issues

At model integration time, the tradeoff between single vs. multiple models becomes most apparent. Separate models mean more work in the integration phase, with possibly less help from automated checking of model semantics provided by the supporting technology. On the other hand, separate modeling efforts can result in better utilization of team resources, smaller, more tractable models, and more potential reuse of models or submodels, both within the current modeling effort and as spin-off results. Each team must determine the mix of these strategies appropriate for their project.

7.2.4.2 Domain Model Interpretation

7.2.4.2.1 Purpose

The Domain Model Interpretation process serves to characterize broad underlying interpretations of the domain that help determine which feature sets and characteristics form viable, cohesive paradigms for system behavior.

Existing systems will cluster functionality and features in certain habitual ways. Some of these ways may be founded on historical precedent or accident; System A implements a given feature similarly to System B because code from System B was leveraged into System A. On the other hand, certain co-occurrences of features may reveal a more essential, logical connection between the features. Domain Model Interpretation attempts to distinguish between feature co-occurrence on this basis.

The Interpreted Domain Model may also identify major divergences in the underlying semantics of domain functions and operations, and may reveal underlying principles that influence many individual design decisions distributed throughout the architecture of systems in the domain. This may result in multiple integrated and/or feature models for a single domain.

The Interpreted Domain Model is still descriptive rather than prescriptive. It serves to explain the co-occurrence of certain clusters of features and design choices within exemplar systems. There may be exemplar systems that violate the feature "affinities" proposed within the Interpreted Domain Model; but these should demonstrate some loss of quality as a result, if the Interpreted Domain Model is valid. It is not intended to encode new design decisions made by the domain modeler team.

7.2.4.2.2 Inputs

Inputs to the Domain Model Interpretation process may include the following:

Integrated Descriptive Domain Models

Interpretation is performed on the descriptive models after integration, so that the interpretation is not obscured by easily caught modeling errors.

Domain Feature Model(s)

Often variant interpretations of domain semantics will become most visible in the construction of the feature model.

Domain Genealogy Model

Genealogical information helps modelers assess the relative importance of perceived commonality and variability in the domain, and to interpret co-occurrence of features in historically related vs. historically unrelated systems.

Domain Interconnection Model

Analogy domains, particularly those related to formerly manual processes that were automated, may prove to be the source of the various interpretations active in the domain.

7.2.4.2.3 Workproducts

Integrated Domain Models

Domain Features Model

7.2.4.2.4 Process

Model Validation

The Interpreted Domain Model is the final phase of descriptive modeling. Validation is done, potentially, at every stage of modeling. Domain model interpretation and innovation steps also perform a validation function. it is appropriate to devote more attention to thorough validation of this last descriptive form of the model, before the model innovation step.

To summarize model validation activities throughout the descriptive phase:

- Individual models are validated according to the conventions of the modeling formalism used, by team review, and by mapping back to actual domain information associated with particular exemplar artifacts.
- Models are integration tested in terms of reconciling any possible overlapping areas of ontology coverage between the models, and for formal consistency across the models.
- The descriptive models are further validated and integrated through the definition of individual features. Each feature ties together specific categories from the conceptual models, is linked to a precedent within some exemplar artifact, to the interest and visibility of particular stakeholders as represented in the Domain Stakeholders Model, and possibly to rationale in the contextual models. Features are thus validated and semantically characterized at a relatively detailed level, while serving to validate the underlying descriptive models as well.
- Individual feature specifications are linked to related variants within descriptive feature models. The various separate feature taxonomies that remain as functional differentiators within the domain proper are themselves integrated during Domain

Model Interpretation. Here, feature variants in different feature taxonomies can be associated together because of a common design rationale, architectural principle or underlying semantic interpretation of the domain. This serves as additional validation for the features and the descriptive models as a whole.

7.2.4.2.5 Sequencing

Domain Model Interpretation can be tightly interwoven with Domain Model Innovation. In particular, the innovation technique of successively repressing features from strongly bound feature sets is one technique for validating the logical rather than merely habitual cohesion of feature sets observed in the described exemplar set.

7.2.4.2.6 Guidelines

Domain feature models will tend towards one of three forms of configuration. If a set of features are logically orthogonal (i.e., separately selectable) the feature set will resemble a lattice; depending on the number of features and feature values, the number of potential variants may be subject to combinatorial explosion. A second scenario occurs when a series of features each depend logically on the previous. In this case, a linear sequence of domain application variants may be defined. Most often, however, feature dependencies will be irregular, ragged sets of relations. It is in these latter kinds of conceptual relationships that representations like semantic networks are actually most useful in keeping track of constraints and semantic relations between terms.

7.2.4.2.7 Issues

Distinguishing between different models built for the descriptive phase and separate domain interpretations that may be identified during Domain Model Interpretation. One rule of thumb to consider is the following: the separate models in descriptive modeling can each refer to artifacts, concepts or contextual information that span the full exemplar set. The domain interpretations, by contrast, will typically partition the Domain Representative Set (and, by implication, the Domain Exemplar Set) into subsets, each of which share a unifying interpretive model of the domain and between which significant differences can be identified.

7.2.4.2.8 Example

In the outliner domain, outlining programs that evolved from an "idea-processor" perspective differed in fundamental ways from programs that treated outlining as an extension of document processing. While both categories are considered outliners, the set of operations supported and even the underlying data structures differ considerably.

Features in the domain of interactive text editors could be partitioned according to those that support a moded vs. a modeless style of operation. While in principle "modedness" itself could be considered a feature, its semantic repercussions, interactions and impact on other features is so pervasive as to warrant its representation as a related ensemble of feature variant choices.

While both of these examples indicate variant interpretations in the domain that are well-known, the ODM method presumes that more complex or specialized domains may reveal such variants only after a certain degree of examination.

7.2.4.3 Domain Model Innovation

7.2.4.3.1 Purpose

The descriptive modeling phase, through the Integrated Domain Features Model, has maintained linkage to precedent in some exemplar artifact. As we prepare to make the transition to prescriptive modeling, we will be making commitments as to which variants of given features assets in the domain asset base ought to support.

Before launching into the prescriptive phase proper, however, it is useful to allow a transitional step, where the requirement that all features described have precedent in existing systems or requirements is loosened, but the requirement to have a clear user context or market for the feature is also, if temporarily, deferred.

This phase in ODM is called the Domain Model Innovation step. It provides an opportunity for innovative design and the discovery of novel opportunities, through combinations of features that have all been proved individually to be attainable. It can help reveal unexpected commonality across systems and even across domains. It also lays the groundwork for a layered design approach that identifies core and peripheral sets of features for system versions within the domain. This layered approach may first be suggested in Domain Model Interpretation if there are several distinct overall semantic interpretations for structure and operations in the domain. It may carry over into prescriptive modeling for the domain architecture as a whole, where the ability to obtain optimized domain variants with the right mix of requirements may be a condition of getting assets reused in practice.

7.2.4.3.2 Inputs

Inputs to the Domain Model Innovation process may include the following:

Descriptive Domain Models

Validated, Integrated, Interpreted Domain Models

Feature Binding Times Model

Domain Stakeholders Model

7.2.4.3.3 Workproducts

Domain Innovation Model

7.2.4.3.4 Process

Domain Model Innovation, by its nature, is not an activity that can be scoped by specific objectives or exit criteria. Practically, innovation activities must be time-based rather than content-based; that is, a certain schedule is created that allows for a period of experimentation, after which prescriptive modeling must proceed. Innovation modeling should therefore not be on the critical path to an adequate domain model and resulting asset specifications. What it can contribute is extra or unanticipated value based on discovering novel combinations of non-novel features.

Rather than offering a systematic linear process for innovation modeling, the approach taken in ODM is to describe a repertoire of specific techniques for model innovation that can be applied opportunistically, based on perceived characteristics of the descriptive models. Some specific techniques are described below.

7.2.4.3.4.1 Feature Orthogonalization

One form of model innovation most directly supported by the descriptive modeling approach described above can be termed *feature orthogonalization*. When features are semantically linked to conceptual models, such as objects, operations and relations models, it is possible to derive novel variants by shifting an existing variant with respect to one of the related models.

For example, suppose that in a domain model for text editors, the operations Delete, Move and Select have been placed in the operations model, and the notion of Character, Word, Sentence, Line, and Paragraph have been included in the objects model as substructure elements of Text. These elements are fully orthogonal: that is, one can delete, move or select to boundaries corresponding to each of the range expressions. Now an Overstrike operation is noted in one exemplar, with the fixed assumption of one character. As part of feature orthogonalization, modelers could investigate whether it is possible to overstrike a word, sentence, etc., in any existing exemplars.

The example show that often innovative modeling ideas will occur at a fine-grained level during descriptive modeling. It is best to capture the ideas informally at this time, perhaps using them as an impetus for further or focused data-gathering, but leaving the resulting models purely descriptive. During Innovative Modeling, these informal notions can be incorporated into an explicit model.

7.2.4.3.4.2 Feature Binding Time Variants

Feature variants will often be associated with particular *binding times* with respect to the various stakeholder contexts identified for the domain. During Model Innovation, feature variants can be shifted in terms of their binding time, to investigate whether other potentially useful combinations will result. For example, it may be possible to link in various algorithms with differing performance characteristics, when a system is configured for a site. During Model Innovation, the possibility of end-user algorithm selection could be considered. The innovation need not always consist of transforming a static to a run-time feature, however. For example, the development of spreadsheet compilers came historically after conventional spreadsheets. Spreadsheet compilers split the schema definition and data entry tasks into separate, independent components of the system, using a generative step after schema definition to produce a data entry program optimized (and hard-coded) for a particular schema. This seeming restriction of functionality actually results in a program configuration more useful than the old one in numerous contexts.

7.2.4.3.4.3 Feature Suppression

During Model Interpretation, regularly co-occurring feature clusters are examined to determine whether the co-occurrence is logically or merely historically based. During Innovative Modeling, individual features can be systematically suppressed from these clusters, to aid in discovering useful "leaner" functional profiles. Suppression may include removal or restriction of a feature. For example, in the text editing domain, the ability to

edit a file in a read-only mode, if not already recognized as a feature, could be discovered by suppressing the modify sub-tree of operations from the profile.

7.2.4.3.4.4 Use of Analogy Domains

During Domain Scoping, various analogy domains for the domain of focus are considered and recorded inthe Domain Interconnection Model. During Model Innovation, these analogy domains can be re-visited for the purpose of finding analogous features supported in one domain but never carried over to the other.

7.2.4.3.5 Sequencing

Innovation Modeling can actually be done concurrently with descriptive modeling; managing this concurrency will, however, require extra discipline on the part of the modeling team. Even if a full-fledged innovation modeling effort is not under way at the same time as descriptive modeling. it can be helpful to anticipate this phase and to save insights and possibilities unearthed during the modeling process in a separate representation from the descriptive models. This can be thought of informally as a "Things We'd Like to See" list, and it can be the source of a number of fruitful ideas for the innovation phase.

One advantage to doing Innovation Modeling after Domain Model Interpretation is that feature clusters can be handled more strategically based on their interpretation. For example, feature clusters that co-occur in exemplar system for purely historical reasons (e.g., leveraged reuse of code among systems) would be ripe candidates for innovation modeling techniques such as methodical subsetting of features. On the other hand, applying such techniques to feature clusters that co-occur because of strong logical cohesion between features may yield more abstract results.

7.2.4.3.6 Guidelines

In creating the Domain Innovation model, consideration of factors such as exact correspondence with the behavioral profile of any existing system, estimated utility of the feature combination or assumed ease of implementation is intentionally deferred. The feature space describable with the domain model may therefore include some seemingly impractical or unusable system variants. These will be culled out after the second scoping effort within prescriptive modeling. The rationale of this approach is to encourage discovery of useful variants that might otherwise be discarded because they do not fit current practice of users or require a shift in anticipated context of use.

The exploration of innovative features will often lead to further data gathering activity, either integrated into the Innovation Modeling step or undertaken in Prescriptive Modeling, during Assetbase Customer Characterization. This data gathering may lead to the discovery that certain novel features are already performed by end-users through some composite set of operations. (Ethnographic study of end-user work processes is another source for this information.) On the other hand, it may be discovered that certain feature combinations were deliberately excluded from system capabilities; the rationale for such feature exclusion may be inadequately documented, and hence only discovered after the possibility is considered and reflected back to domain informants.

7.2.4.3.7 Issues

One risk of this approach to innovative feature modeling involves the potential combinatorial explosion of variants within the model. Aspects of the ODM approach that provide some control over this complexity of feature combinations include:

- The extra care taken in defining domain boundaries;
- Isolating feature subsets of the problem domain as subdomains that are treated as "black box" elements within the feature model;
- Emphasizing relatively small domains, with manageable numbers of features but richly modeled semantic relationships; and
- Use of a modeling formalism that allows for multiple inheritance, documentation of constraints and selection of aggregate configurations.

7.3 Prescriptive Domain/Asset Base Modeling

This section describes the ODM Prescriptive Modeling Phase. The overall structure of the processes in this phase are depicted in Figure O. The following subsection provides a general introduction to prescriptive modeling within the context of ODM, characterizing it and contrasting it to the descriptive and implementation modeling phases. Subsequent subsections describe the three major process subtrees within the prescriptive phase, as depicted in Figure O. Setting the asset-base context is a context-setting phase analogous in many respects to the initial context-setting processes within the descriptive phase. In prescriptive modeling, this phase allows for selection of intended customers for the asset base. Architecting the domain could similarly be compared to the central domain modeling processes of the descriptive phase; in the prescriptive phase, this process involves selection of the architectural and feature variants to be supported in the asset base, and the development of an architectural framework that will allow these variants to be derived and maintained in an integrated fashion. The Architecture Refinement processes are similarly analogous to the Domain Model Refinement stage, involving integration and validation of the architecture, and the exploration of alternative implementation strategies for the assets themselves. These three major processes of the prescriptive phase can also be seen as specializations of the CFRP's general Reuse Management idiom (Planning, Enactment and Learning process families).

7.3.1 Introduction

This section provides an introduction to the prescriptive modeling as a whole. It outlines the purpose of prescriptive modeling, an overview of the process, and details some distinctions between the descriptive and prescriptive phases within the ODM context.

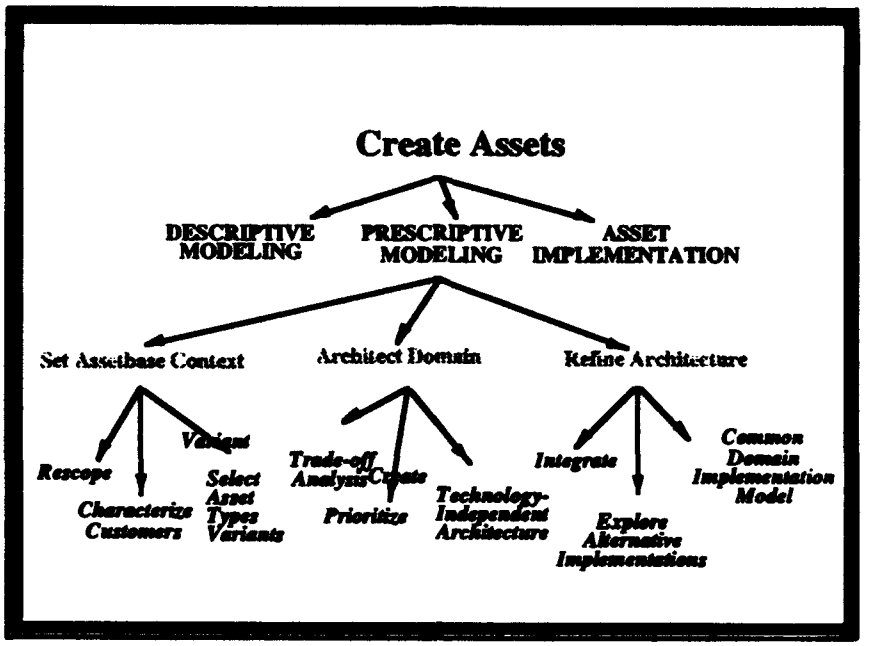


Figure O: ODM Prescriptive Modelling Process Tree

7.3.1.1 Purpose

The purpose of prescriptive modeling is to determine the scope of applicability for the asset base to be developed for the domain, specify the range of feature coverage to be provided by the asset base, and develop specifications suitable for asset implementors to create the assets for this asset base.

7.3.1.2 Overview of the Prescriptive Process

The prescriptive model is produced from the Domain Innovation Model via a convergence process, where both the scope of potential scope of applicability and features to be supported may be modified. The driving force on convergence is the identification of customer contexts to support identified feature combinations (including features that are in effect relaxations of requirements) that are candidates for inclusion in the prescriptive model.

In descriptive modeling, we developed feature profiles and profiles of existing architectures (if available) for representative exemplars within the domain. Each feature in the descriptive model was traceable to a precedent within the exemplar set. In innovation modeling, the transition activity between descriptive and prescriptive modeling, we considered novel feature combinations, the projection of features to unprecedented binding times, the suggestion of new features by reference to relevant analogy domains, and a variety of other techniques for generating innovative but constrained variations on features and architectures.

In prescriptive modeling we begin to make binding commitments on the range of functionality to be provided by assets developed/adapted for the asset base. We need not commit to all feature combinations noted during descriptive modeling. Some of these may not be of sufficient general use to warrant inclusion in reusable assets. Other functions,

possibly of great importance for reuse, may not have been implemented in any existing system, or specified in any known set of requirements for new systems.

Although we do not want to be unduly bound to the descriptive model, we could not have got to the prescriptive phase without, implicitly or explicitly, going through a descriptive phase first. Designers who claim they started with the prescriptive model have simply allowed the concepts and biases they derived from induction and previous experience to remain tacit and informal in the design process.

We have used the innovative modeling phase as a transition to the prescriptive phase. However, we also do not want to be bound to all the potential variants explored in the Domain Innovation Model. In fact, in order for this modeling phase to have the necessary quality of brainstorming, experimentation and rapid prototyping, it is important that developers know they are not making commitments by adding features to this model.

7.3.1.3 Distinctions between Descriptive & Prescriptive Phases

The Prescriptive Modeling phase serves as the transition from the descriptive phase, where existing systems and known requirements are studied comparatively, to the Asset Implementation phase where specific reusable assets are developed and/or re-engineered. Descriptive domain models describe existing systems or system requirements in the domain . The Interpreted Domain Model separates intuitively cohesive feature combinations based on global principles. However, it still offers no prioritization of variants , no assumptions about functional capabilities or implementation strategies for assets. The Domain Innovative Model introduced new and restricted feature variant combinations, but each of these represents only a **potential** asset capability. The prescriptive modeling phase moves us from the realm of possibilities to commitments.

Informants to Customers. The descriptive phase gathers domain data from informants. There is no implicit commitment to provide assets that would serve the needs of any particular context of use studied in the descriptive phase. Deployed legacy systems may be studied that have been in use for some time; back-filling components into these maintained systems may be quite difficult, or might create constraints on implemented assets that would frustrate their use in other, more critical contexts. Nevertheless, the study of functionality in these uncommitted contexts of use may be quite valuable in deriving a complete and context-reflective descriptive model; in fact, knowledge that we may study exemplars we may not finally support allows the freedom to model from a rich diversity of sources of information. In context-setting, we determine specific sets of customers to be supported (in the context of reuse, application developers who are targeted utilizers of the assets to be developed).

Difference in Scoping Activities. The parallelism of structure between descriptive and prescriptive phases may be somewhat misleading, because the basic process being performed is quite different. In descriptive modeling, we built models, possibly using starter models available in the infrastructure as resources, but primarily through analysis and interpretation of exemplar artifacts. Later stages of descriptive modeling involve direct transformation operations on the models themselves. In prescriptive scoping, the defined descriptive scope is modified (hence the term rescoping) to reflect binding commitments for the asset base market. We also do scoping on the models themselves, identifying

possible subsets of the models that will become the starting points for the prescriptive models.

For example, in the outlining domain, Domain Model Interpretation yields the insight that two fundamental outlining paradigms exist within the domain exemplars, document-based and idea processor-based outlining. In the prescriptive phase, modelers might decide to address only one of these interpretations within the asset base. The descriptive model still retains its own intrinsic value, and could be revisited during subsequent prescriptive phases for the same domain, or for prescriptive phases of ODM projects for related domains.

Difference in Model Evolution Activities. Once in the modeling process itself, the descriptive and prescriptive phases also proceed quite differently. In fact, the overall Descriptive - Prescriptive - Implementive process model can be characterized as a "mirror-image" structure, where artifacts are abstracted into features, and features are re-structured back into reusable artifacts, or assets. In prescriptive modeling, therefore, we start from feature descriptions and build forward links to possible contexts of use, relying on retained linkage to descriptive exemplars for existing features but allowing for discovery and exploration of new contexts of use for innovative features.

Transition from Prescriptive to Implementive Modeling. The Learning phase of Prescriptive Modeling leads into the Planning phase for Asset Implementation. However, the border between these two activities is more difficult to prescribe in advance than the transition from descriptive to prescriptive modeling. Essentially, the relationship between Prescriptive and Implementive modeling is close to the notion of requirements to design in conventional software engineering. The prescriptive model says what features must be supported by the asset base; the implementive model embeds specific choices of technology that determine how the asset base components, generator and architecture will satisfy these feature requirements.

7.3.1.4 Sequencing

Prescriptive modeling is assumed to take place after descriptive modeling, and to be able to draw on the workproducts created by descriptive modeling. Alternative strategies for project management of the overall Asset Creation processes can be considered, including parallel, pipelined, or iterative organization of descriptive and prescriptive modeling. In any of these cases, however, a core principle of ODM is to maintain clarity about whether descriptive or prescriptive modeling is being performed, and to keep the resulting models separate.

7.3.2 Set Asset Base Context (Plan)

The following process descriptions detail activities in the Context-Setting phase for prescriptive domain modeling. Parallels with phases and activities in descriptive modeling are pointed out, where relevant. Closer parallelism than indicated here is possible, at the discretion of the project team. One approach would be to address each workproduct from the descriptive context-setting phase in turn, modifying it as appropriate for the rescoped domain boundary. For example, the Domain Definition Statement could be rewritten into an Asset Base Definition, to indicate the intended scope of the asset base itself.

7.3.2.1 Define Asset Base Customers (Rescope)

7.3.2.1.1 Purpose

As the first planning step in Prescriptive Domain modeling, we need to reassess the proposed scope of applicability for the proposed asset base we intend to create for the domain of focus.

This rescoping provides a baseline for evaluating and prioritizing the potential usability of the feature variants derived from descriptive and innovation modeling.

We work from the innovation model, and from characterizations of customer contexts that were examined (if only indirectly in some cases) during descriptive modeling. Some contexts will be excluded from active consideration; for example, some legacy systems may be worth studying in terms of descriptive features, but are not viable candidates to reengineer for reuse. New aspects of existing markets may also be studied. These include possible undocumented needs in this developers' or end-users' environments, suggested by particular feature innovations. Finally, we may look for entirely new markets. In order to keep this process relatively bounded, two particular techniques of market discovery can be applied: first, looking for related markets in terms of the CFRP-derived Domain Stakeholders Model (i.e., find pre- and after-markets, value-added resellers etc.); second, look for market extensions based on the domain relations described in the Domain Interconnection Model.

7.3.2.1.2 Inputs

Inputs to the Define Asset Base Customers process may include the following:

Integrated and Interpreted Descriptive Domain Models

Innovation Domain Model (kept distinct from the descriptive models)

Stakeholder Relations Model: output from Organization Assessment process, used to discover related markets in CFRP terms. This model will be refined and updated to document new markets discovered as a result of model innovation and the rescoping activity at the start of prescriptive modeling.

Domain Interconnection Model: used to explore possible extended markets for innovative features or feature restrictions.

7.3.2.1.3 Workproducts

Modified Domain Stakeholders Model

Domain Asset Base Intended Scope

7.3.2.1.4 Process

The Domain Exemplar Set and Domain Representative Set were selected in order to provide an adequate range of diversity for descriptive modeling. Inclusion of a feature within the descriptive model was legitimate, even on the basis of a single precedent within the exemplar set.

We will go through a similar set of steps for Prescriptive Modeling, with some significant differences in intent. Here, we want to be clear who our customers are. Our initial specification of customers for the asset base will be definitional in nature; we need not propose to do thorough "market research" on all these customers. Just as in descriptive modeling, we will select a representative set of customer contexts for prescriptive modeling. An important criterion of this set is that the number of potential reuses for an asset specification, as characterized relative to these customer contexts, will be as indicative as possible about the relative market for that asset within the customer base as a whole. In other words, we would like to find a typical rather than (or in addition to) a spanning set within the potential marketplace. We will prioritize feature profiles for assets, in part, not with respect to a single context of potential use but the estimated frequency and distribution of these uses within the marketplace.

As a result of the rescoping decisions made in this phase, it may be necessary to update or revise the domain definition and domain boundary decision workproducts. If the descriptive models are intended for future use in their own right, however, it may be more appropriate to create separate definition and scoping workproducts for the prescriptive domain. For example, the descriptive models may remain as aids to program understanding for certain deployed systems that still require maintenance and evolution, but that are not targeted as recipients of reengineered, reusable components from domain engineering.

7.3.2.1.5 Sequencing

There is a risk that iteration between innovation modeling and rescoping of the committed customer context will result in an out-of-control process; i.e., an innovative feature is conceived, a potential customer is found, the customer context is studied, opportunities for other features are discovered, the innovation process continues, and so forth. In this case the set of features and the proposed customer contexts could continue to spiral out of scope. One risk mitigation strategy is to preserve the sequence of descriptive, innovative, and prescriptive modeling, and to limit the feedback cycles allowed. In this way the descriptive modeling phase puts an empirical boundary on the process. Since legacy systems will be an important source of prototype components for the asset base, this approach makes good sense since it reflects the resources that will be applied.

7.3.2.1.6 Guidelines

Criteria for scoping

Some contexts studied during descriptive modeling may be excluded from the prescriptive customer base. For example, if older legacy systems were studied during descriptive modeling, but there will be little chance or motivation to backfill newly developed reusable components into such systems, they may be excluded from the planned customer base.

On the other hand, new potential customer contexts may be discovered by exploring potential uses for asset features. This can include both fully precedented features (i.e., part of the descriptive model proper) or feature combinations derived from Innovation Modeling. For example, a feature available to developers in exemplar systems may turn out to be a useful feature to provide to system end-users. By methodically considering the impact of

migrating features across the feature binding time spectrum for the domain, such previously unanticipated enhancements can be discovered.

Here the line with innovation modeling can get blurry. Migrating a given feature across the feature binding time spectrum is, in and of itself, a transformation that could be applied during innovation modeling. Once applied, and used to generate a new feature variant, however, it falls to the Context-Setting phase of Prescriptive Modeling to actively search for candidate contexts of use for this new feature. In these cases, novel features suggest new possible contexts of use.

7.3.2.1.7 Issues

If the prescriptive customer base can either expand or contract, relative to the descriptive exemplar set of contexts, what validation is possible between these two scopes of applicability?

7.3.2.2 Characterize Asset Base Customers

7.3.2.2.1 Purpose

The Asset Base Customer Characterization process provides the supplementary information about the intended contexts of use for reusable components to guide variant trade-off and asset implementation decisions. Although we have identified "customer contexts", prescriptive modeling does **not** represent pure market analysis. In particular, this process does not involve thorough analysis of the spectrum of needs within these contexts. Domain engineers are definitely in the position of technologists with something specific to sell—namely, the domain functionality modeled to this point. We need not even provide *all* functionality for the domain required in these contexts. The asset base must provide enough domain coverage that it will be worth the effort for application engineers to utilize the asset base in developing systems. The boundary of what is and is not supported in the asset base should be clear enough that potential asset utilizers do not seek in vain for assets that are not there, and do not neglect to search for needed assets that are there.

7.3.2.2.2 Inputs

Inputs to the Customer Characterization process may include the following:

Modified Domain Stakeholders Model (from Define Asset Base Customers process)

Domain Asset Base Intended Scope (from Define Asset Base Customers process)

Domain Innovation Model

Organization and Technology Assessment Results

7.3.2.2.3 Workproducts

Asset Base Customer Characterization

7.3.2.2.4 Process

The following paragraphs discuss some suggested activities for this process.

First, some supplementary characterization may be needed for potential contexts of use not originally examined during descriptive modeling. These would include contexts identified as potential contexts of use for innovative feature combinations developed during Innovative Modeling.

For example, suppose that in innovative modeling for the database domain, a variant was defined that strips away all non-essential database reporting and ad hoc querying procedures, preserving only the core functionality of data storage and management and compiled queries. There may be a large set of potential contexts of use that currently use custom-built data management systems that could be potential utilizers of such a "lean database" variant. However, this class of applications may not have been considered during the initial descriptive modeling stage.

Second, even contexts of use that were examined during the descriptive phase may need to be re-examined from the standpoint of novel feature variants explored during Innovative Modeling. For features that are shifted to an alternative binding time, for example, additional characterization of the workflow in the stakeholder contexts might be needed in order to determine how useful such a feature would be. Such information might very likely not have emerged from the initial domain information gathering, since in effect it concerns potential and not existing features. It is possible that user requests for enhancements or modifications may have anticipated such innovative features, however. It is often the case that developers provide ancillary functions for their own environment that would be useful to end-users as well. (By exploring such innovations, modelers may discover that this functionality has been withheld from end-users for definite and deliberate reasons. This would be a prime example of the kind of rationale that would appear in contextual models of the domain, and would affect how such features or lack of features would be modeled.)

Third, all the potential customer contexts must be characterized more fully with respect to the actual implementation forms and packaging of the reusable assets to be provided. Up until this point, the focus of the domain modeling process could be said to be primarily on the functionality provided within the domain. However, some knowledge of the current forms of workproducts (studied as artifacts) in various domain contexts has been captured in the artifacts models during Descriptive Modeling. These models can now be reexamined and refined. The usage of the artifact models shifts at this point; initially they served as a way of providing traceability from conceptual model entities back to precedents within domain exemplar artifacts. Now these models must express constraints on the actual implemented form for assets within the domain.

For example, assume that a number of requirements specifications, following a particular standard, were examined during descriptive modeling. The artifact models constructed at that time noted certain aspects of these requirements documents, including their internal structure, sufficient to allow specific requirements within these documents to be referenced as source material for various domain terms and feature descriptions. Now, in prescriptive modeling, we must consider whether the asset base will itself contain requirements text and, if so, requirements documents (or document templates, fragments, generators, etc.). If so, we will probably require a finer level of detail about the use of such documents within the various customer contexts.

As a result of this process, the Domain Stakeholders Model may be further refined to reflect additional contexts of use and additional fine-grained life-cycle stages within these contexts of use.

7.3.2.2.5 Sequencing

It is appropriate to defer detailed characterization of customer context until after some initial choices of these contexts have been made, simply as a means of filtering complexity and conserving resources. Hence Asset Base Customer Definition and Characterization have been presented as sequential processes, with characterization following definition. In practice, of course, these processes may be highly iterative, and may in fact bear closer resemblance to the process followed in original domains selection: identification of candidates, characterization of the candidates to a degree of general interest within the business area, followed by a more focused selection process.

Similarly, the process model outlined here implies that final selection of asset types to be supported can follow the characterization of customer contexts. In practice, there may be mutual dependencies between these processes, and some parallel and/or iterative performance will be necessary.

In some approaches to domain analysis, the characterization of customer contexts outlined here could be considered part of a process related to, but somewhat independent of domain analysis proper. For example, Patricia Collins (HP Corporate Engineering) describes a process termed "reusability analysis" in [Col92] that involves many of the elements described above. In this process model, this analysis would take place for earlier in the process than described in this technical report. However, this model in part reflects an assumption that the set of customer contexts to be supported is given, and not a function of the domain selection and modeling process. If this is the case, this obviously provides a point of process stability not assumed in the ODM process model presented here.

7.3.2.3 Determine Asset Base Coverage

7.3.2.3.1 Purpose

In this final process of context-setting for the Prescriptive Modeling phase, additional constraints on the scope and comprehensiveness of the asset base are established. Based on the customer contexts defined and characterized in the previous processes, the types of assets to be supported in the asset base, and the overall architectural principles for integrating and composing these assets is determined in this process. It is therefore closely analogous to the Determine Domain Coverage process in the Descriptive Modeling phase.

7.3.2.3.2 Inputs

Inputs to the Determine Assat Base Coverage process may include the following:

Domain Information Types Model: provides a cross-check for completeness in terms of the types of artifacts selected to be supported as assets.

Domain Descriptive Models: in particular, the artifact models created in the initial descriptive phase. It is now important to know the structure of existing artifacts that may be candidate asset typss, since assets may need to be provided with similar structure. This does

not imply that asset implementors will need to develop the assets directly following this structure. The structures themselves may be modified, and alternative techniques such as generative techniques could be employed to produce the workproducts in the formats needed.

Customer Characterizations: in particular, knowledge about the potential contexts of use is required to determine what asset types are most likely to be used by asset utilizers, and what architectural formalisms will be accessible, understandable and usable by them.

7.3.2.3.3 Workproducts

List of Asset Types to be supported by the Asset Base.

One or more architectural formalisms to be employed in defining the domain/asset base architecture.

7.3.2.3.4 Process

The following paragraphs describe constituent activities of this process.

7.3.2.3.4.1 Select Asset Types

This step is analogous to the Select Domain Information Types step (within the Determine Domain Coverage step, Context-Setting, Descriptive Modeling), where now the workproduct types selected represent the types of assets that will be supported in the asset base rather than the sources from which descriptive information will be gathered. In selecting the asset types to be supported, modelers are, first and foremost, identifying the life cycle phases to be supported with reusable assets. For example, if only requirements-level assets are to developed in this cycle of asset creation, this decision would be reflected in the exclusion of code assets from the list of selected asset types.

7.3.2.3.4.2 Define and Scope Architecture Formalism(s)

This step is analogous to the Select Modeling Formalisms step (within the Determine Domain Coverage step, Context-Setting, Descriptive Modeling), where now the representation formalism(s) to be selected address architecture description rather than descriptive/comparative modeling per se.

In principle multiple, heterogeneous architecture formalisms could be selected in this phase. However, since the architectural view allows the specification of relationships and aggregate structures among individual components, there will be an obvious disadvantage in trying to integrate individual assets or sub-architectures designed on the basis of different formalisms.

One dimension in which alternative architectures may be appropriate would be in cross life-cycle assets. In the sense intended by the ODM method, where a domain architecture addresses strategies for interconnection of individual assets and ensembles into a coherent overall structure, there will be as much need for "requirements architectures" as conventional architectures in the sense of module topologies. (Note: this is a very unconventional use of the term "architecture".)

For example, individual requirements (represented, for example, in textual form) could be understood to be a non-code asset (capable of being reused in future requirements specifications, and possibly encoding a great deal of domain knowledge in its form). If utilizers are expected to be able to derive complete and consistent requirements specifications for systems in the domain from the asset base, the asset base will need to produce not only individual requirements but related sets of requirements and even entire requirements documents, tailored appropriately. This set of requirements can be considered an "architecture" of sorts, and the asset base will need to represent relations between requirements in such an architecture, just as it will need to represent module interdependencies (but obviously, not using the same architectural formalism).

7.3.2.3.5 Sequencing

When rescoping results in a requirement for new types of assets or new customer contexts, some additional descriptive study may be necessary.

7.3.2.3.6 Guidelines

There does not need to be exact correspondence between the workproduct types studied in descriptive modeling and the asset types selected for engineering in the prescriptive phase. One advantage of specifying the same types is that the chances for reengineering existing artifacts as part of asset implementation are clearly greater if the same types of workproducts are being produced. For example, if a code module supporting a given set of features is to be produced in asset implementation, having access to analogous modules from legacy systems that implemented subsets or supersets of the features required for the asset would be useful leverage in the asset implementation phase.

Another advantage of specifying common artifact and asset types is that more information about the developer's requirements for given assets will be known for those studied descriptively. For example, if test data artifacts were studied during descriptive modeling, and modelers decide to include test data as a form of asset in the asset base, the descriptive model should contain some information, not only about the content of various test data sets relevant to domain functionality, but also about the required format, organization, and documentation of test data, given the software development methods in use and local programmers' culture.

However, artifacts from any phase of the life cycle may contribute information useful in reengineering assets from any other part of the life cycle. So, for example, the study of test data in the descriptive phase may provide essential information for the engineering of reusable designs in the asset base.

7.3.3 Create Prescriptive Models (Enact)

The following sections describe the main processes in creating the prescriptive models for the domain. The Prescriptive Context-Setting phase has identified potential asset customer contexts and characterized these contexts to facilitate trade-off analysis of feature variants and asset implementation strategies. Usability and Feasibility Analysis processes evaluate potential feature variants against these customer contexts, to determine which variants will be useful and feasible to implement. This trade-off analysis is performed from both a bottom-up and top-down perspective, focusing alternately on specific functional features to be supported and on overall architectural approaches for domain functionality. Variant

functional and architectural features are prioritized according to usability and feasibility criteris, and a subset of these features are selected for support within the asset base, based on available resources and overall objectives.

In the final stage of prescriptive model evolution, committed features are allocated to a technology-neutral *domain architecture* . This architecture includes specifications for *asset ensembles,* functional areas of the prospective asset base to be implemented with separate components, generative techniques, or hybrid strategies. The architecture also specifies the high-level interactions across these asset ensembles. In the Asset Implementation stage that follows Prescriptive Modeling, these ensembles will be further decomposed into individual asset specifications and alternative implementation strategies.

7.3.3.1 Usability Analysis

7.3.3.1.1 Purpose

The Domain Innovation Model has produced some novel feature variants, intermixed with variants that map to exemplars or known requirements. During usability analysis, the feature combinations in the innovative model are used for an exploratory search of potential customers for assets with the features described.

7.3.3.1.2 Inputs

Inputs to the Usability Analysis process may include the following:

Domain Innovation Model (from Innovative Modeling process);

Refined Stakeholder Model;

Characterizations of Contexts of Use;

Selected Asset Types;

7.3.3.1.3 Workproducts

Variant Feature Usability Analysis

Variant Architecture Usability Analysis

7.3.3.1.4 Process

The following paragraphs describe the primary activities for this process.

7.3.3.1.4.1 Bottom-Up Vs. Top-Down Analysis

Usability analysis is performed as an iteration between bottom-up and top-down analysis. In bottom-up analysis, individual features and feature variants are investigated for possible application. In top-down analysis, feature profiles for entire domain systems (that is, systems or subsystems representing the full scope of the domain of focus, rather than individual features within the domain) are examined from the perspective of usability in different contexts.

Descriptive and innovative domain models have translated domain functionality into sets of features and feature variants. There may be many, if effect, sub-models describing the variants examined for one particular feature, e.g., one functional capability in its many forms as observed across the exemplars. In addition, there may be composite descriptions of *feature sets* that represent full functional profiles for various versions of the domain system or subsystem as a whole. Within this context, these feature sets can be thought of as describing *architectural variants* for the domain; that is, each describes a particular configuration that, if committed to as part of the prescriptive model, must be obtainable as a supported system configuration from the final asset base.

Note that, in the context of this activity, the term *architectural* is used to connote a domain-wide scope; this does not always imply a particular design or system topology. Original structure of the artifacts from exemplar systems is preserved only insofar as that structure places constraints on the desired assets. These constraints may be expressed as features linked to the developer's context of use; e.g., a feature of a code module may be that it is implemented in C++, a feature of a requirements document may be that it complies with a government documentation standard. These constraints can be preserved in the feature model, along with features representing functional requirements for systems in the domain. Thus, an architectural variant is best thought of as a set of features describing the functionality of a full system variant for the domain.

7.3.3.1.4.2 Feature-Driven Vs. Context-Driven Analysis

The process can also be viewed as iteration in another sense: between "feature-driven" and "context of use-driven" analysis. Just as descriptive modeling represented a strategic choice between domain data- and ontology-driven modeling, here the flow of activity can be structured by the variant features themselves, each one initiating an exploration of new contexts, or by focusing on a new potential context of use for domain assets and exploring possible needed features in that context. In general, because of the emphasis on producing tightly constrained feature models in the earlier processes in ODM, the former strategy is best used as the primary structuring framework for the process.

7.3.3.1.4.3 Quantitative use estimate

Simply finding a context of use is a way of demonstrating the usability of a given feature variant, in principle, but certainly not of estimating its usefulness relative to other possible variants. The notion of variant prioritization here is based on a fundamental principle that support of variability always carries a price, whether in performance, asset base overhead, complexity of the asset search and retrieval process, or increased complexity in configuration management and asset maintenance. Therefore, there needs to be *sufficient* potential use for a feature variant to warrant its inclusion in the feature profile for an asset.

One approach to this prioritization is to attempt to estimate the potential "market share" for a feature variant within the intended context of use. This quantitative analysis of reuse potential is an important output of the DA process, and has been emphasized in several other DA techniques [???].

However, it is important to keep in mind the limitations of our current methods in generating this quantitative estimate. For example, because reusable assets might employ

generative as well as component-based techniques, the potential contexts of application might not be readily apparent from either an informal survey or code-based analysis.

For example, to compare two feature variants with respect to their reuse potential, one approach would be the following:

- Across instances of use, come up with estimated scale of cost-to-adapt per instance of use;
- Characterize each variant across its instances of use using cost-to-adapt estimate;
- Allow for instances excluded or enabled by a given variant as well as straight comparison.

7.3.3.1.5 Sequencing

Usability and feasibility analysis can be done in parallel, as can the bottom-up and top-down aspects of each analysis process. Various process models can therefore be defined for variant trade-off analysis as a whole.

7.3.3.1.6 Guidelines

Although we are specifically *not* committing to an implementation method at this point, it can be helpful to consider some alternative implementation strategies in the context of feature variant prioritization.

One option is simply to select a single variant as the variant to be supported in the asset base. In this case, the other variants are excluded from the prescriptive model and the asset implementors need not address how to obtain those variants.

Another option is to implement both variants, even if both could not co-exist in any single run-time system. In this case, the asset implementor might develop both variants and store them in the asset base, or implement a generator that can create both variants as instances, to be generated on request by asset utilizers or generated and archived.

7.3.3.2 Feasibility Analysis

7.3.3.2.1 Purpose

The various feature combinations and architectural variants are also analyzed from the point of view of feasibility. This includes the ease of implementation (if possible, avoiding pre-emptive commitment to particular implementation techniques, however) and the difficulty of maintaining separate variants within the same asset base. In a sense, this involves a packaging tradeoff analogous to that in product family planning, ensuring that particular feature clusters have sufficient breathing room across the intended range of customers to warrant the overhead of separate creation and configuration management.

7.3.3.2.2 Inputs

Inputs to the Feasibility Analysis process may include the following:

Domain Innovation Model (from Innovative Modeling process);

Refined Stakeholder Model;

Characterizations of Contexts of Use;

Selected Asset Types;

Technology Assessment results;

Domain Modeling Infrastructure Plan (and Asset Management Infrastructure Plan, if available from outside the context of the ODM project proper).

7.3.3.2.3 Workproducts

Feasibility Analysis for Domain Feature Variants

7.3.3.2.4 Process

The following paragraphs briefly describe suggested activities for this process.

7.3.3.2.4.1 Implementation

For each feature variant to be considered, the relative cost/level of effort for implementation for the variant is considered. For variants that have been implemented in previous systems, the cost of implementation can be presumed to be, at least, more predictable than for novel features. If project metrics are available for exemplar system development efforts, levels of effort for initial implementation, and later maintentance of artifacts implementing desired features may provide some guidelines. Clearly, however, such estimates will be far from an exact science.

For novel feature variants, the estimates will be more difficult. Here, it may be necessary to examine existing artifacts and to project the reengineering or restructuring required to produce the required variant. Additional input from domain experts will probably be necessary for this process.

7.3.3.2.4.2 Maintainability

In addition to estimating levels of effort for initial implementation of the feature variants, the relative cost and/or level of effort to maintain the variants should be considered. This reflects a general rule in designing for reuse: the reusability of each asset must be considered within the context of the asset base within which it will reside, and not only in isolation.

Configuration management issues should also be considered as part of maintainability issues. Here, the problem of providing separate variants can be compared to the configuration management problem of a system where multiple versions remain in the field simultaneously and must be managed in parallel. Since such configurations can become configuration management nightmares for even a single system, these issues should be carefully considered in the context of an asset base of multiple reusable components.

7.3.3.2.5 Sequencing

Feasibility analysis (covering implementation and maintainability) will need to be revisited after asset implementation choices have been made. For example, three closely related variants may appear to necessitate three independent and separately maintained code components; however, if generative techniques are used during asset implementation, the three variants may be derivable (and maintainable) from a single source workproduct.

7.3.3.2.6 Example

In the design of the Reusability Library Framework, several variants of the semantic network component of the system (AdaKNET) were considered, including variants that provided only specialization relations (no aggregation relations), only single and not multiple inheritance, etc. Potential contexts of use for each variant were conceived; however, the effort required to maintain separate builds of the system for each of these variants proved prohibitive, given the overall project objectives and resources available. The defined variants still proved useful as part of the Phased Implementation Plan (described in Section 7.4.2 below).

7.3.3.3 Evolve Prescriptive Models

Previous activities have characterized feature variants and architectural variants according to potential usability and feasibility of implementation and maintainence. In the prescriptive modeling evolution process, binding commitments are now made to the features and architectural variants to be supported.

7.3.3.3.1 Purpose

The purpose of prescriptive modeling evolution is to derive an overall profile for an asset base that will have the strongest possibility of being utilized on a consistent basis by developers in the domain, and that will provide the highest level of support to these developers.

7.3.3.3.2 Inputs

Inputs to the Prescriptive Model Evolution process may include the following:

Innovative and Descriptive Feature Models

Selected Asset Types and Architecture Formalism(s)

Usability Analysis Results

Feasibility Analysis Results

7.3.3.3.3 Workproducts

Modified Asset Base Stakeholders' Model

Domain Prescriptive Feature Model(s): Contain all feature variants committed to be supported within the asset base. Each variant is linked to prospective contexts of use within the Modified Asset Base Stakeholders' Model.

Domain Prescriptive Architectural Model: Contains a set of asset ensemble specifications, each allocated a set of feature variants, with high-level interactions specified between the ensembles within an overall architectural framework. In particular, this framework specifies specific architectural instantiations that can be derived for the domain from the common architectural framework.

7.3.3.3.4 Process

The activities within this process are described in the paragraphs below.

7.3.3.3.4.1 Prioritize Feature and Architectural Variants

Results of the exploratory analysis are synthesized into a model that represents the binding commitments or specification for the functional profile of domain assets. Some contexts within the descriptive scope may be *de-committed,* because of insufficient need for functionality required in that context across the domain scope as a whole. Other customer contexts might be introduced at this time that will require more detailed characterization. Contexts of use are not necessarily excluded through "negative evidence", i.e., because no features have been allocated to them. This may be a result of the selective nature of descriptive modeling, which does not necessarily identify every context in which a given feature would be needed or useful, but only a typifying subset enabling comparative modeling to be performed.

7.3.3.3.4.2 Allocate Variants to Asset Ensembles

Once the variants to be supported by the asset base have been selected from the set of variants discovered through descriptive and innovative modeling, a high-level clustering of the selected variants is performed that allocates groupings of these variants to particular groupings. The output of this phase is an overall structure for the asset base, including a range of architectural variants to be supported by the asset infrastructure and a set of specifications for "asset ensembles".

Asset ensembles represent functional areas within the domain that may be implemented with sets of static components, generative tools, or hybrid solutions; and which may be developed using a combination of newly developed workproducts and adaptations of existing artifacts characterized during the Descriptive Modeling phase. The architecture and asset base structure produced during the Prescriptive phase attempts to structure these ensemble levels so that shifts in asset implementation technology will have localized impact on the asset base structure as a whole.

7.3.3.3.4.3 Consider Alternative Architectures

In this step, various architectural approaches are considered to accommodate the specific feature and (system) architectural variants committed to for the prescriptive model. ODM assumes the eventual development of a *repertoire* of architectural models, available via a handbook, catalogue or asset base. This is in keeping with the notions of architectural idioms suggested by some researchers (e.g., [Gar93]). The architectural modeling step can therefore involve perusal of this architectural repertoire and experimentation with alternative approaches, each with accompanying support infrastructure. Each architectural model built for a domain is engineered with the expectation that it may find its way into such a handbook as part of its refinement and generalization.

Note that the "architectures" considered here are essentially ways of structuring the set of asset ensemble specifications that, together, make up the functional scope for the asset base to be implemented. Specific system architectures within the domain must be derivable via instantiation from the domain architecture, that is, from implementations within asset ensembles, combined and composed according to constraints known to the domain modelers.

7.3.3.3.4.4 Determine Asset Base Architecture

After variant architectures have been considered, a selection is made that best satisfies the multiple constraints imposed by the various contexts of potential use. One primary tradeoff will be between closeness of fit to particular exemplars (or, at this point, customer contexts) versus the flexibility to be used by multiple customers.

To the extent that ease of integration with one existing system as an asset base customer is of high priority, the stronger will be the pressure to adopt an architectural approach compatible with that exemplar system. One way to avoid undue influence of one existing architecture is to link a domain engineering effort with a reengineering/restructuring effort for the system in question. This allows the consideration of alternative architectures for the system, with the effort involved in restructuring amortized beyond the asset integration task itself.

7.3.3.3.5 Sequencing

Commitments may in fact be made in interleaved fashion during the prior analysis steps. The process model here suggests only that the commitment decisions be tracked separately from the linkage to potential customer contexts etc. One reason for this is that the linkage information may prove valuable in the future, if functionality that was excluded from one prescriptive domain model/architecture/asset base may become a candidate for inclusion in a related domain engineering effort.

7.3.3.3.6 Guidelines

7.3.3.3.6.1 ODM Concept of Architecture

In order to clarify the architectural decisions made at this point in the process, it is critical to understand the sense in which the term "architecture" is employed in the ODM context.

The ODM asset base includes the assets (components, generators, etc.) developed in the asset implementation stage, as well as making available on an ongoing basis the various domain models created during descriptive and prescriptive modeling. The ODM asset base can also include specific architectural models for systems (or subsystems) in the domain as components in their own right.

The asset base itself does not impose one particular system architecture that must be adopted as a standard by all applications making use of the assets. More typically, the asset base will provide a configurable architecture framework, from which a class of specific architectures can be derived by asset utilizers.

This class of architectures could be "implemented" in the final asset base in a variety of ways, like other assets: as a collection of specific architectures, retrieved upon request; as a

set of sub-architecture pieces that can be assembled in various ways; as an architecture description language (required only to cover architectures typical of systems in the specific domain) that can be used to generate an instantiated architecture for a particular system, based on an input specification, etc.

This "space of possible architectures" can be considered the *domain architectural model.* In a domain where an architectural approach makes sense (i.e., a domain with aggregation and substructure in target systems, as opposed to a flat set of components, such as math routines) the architectural model becomes the "skeleton" or "spine" of the asset base. It would structure the set and sequence of choices a utilizer would make in instantiating an individual system architecture from the asset base.

One possible point of confusion is the fact that an architectural model of this kind could, in theory, be developed in the descriptive modeling stage as well. However, a descriptive architectural model would cover the range of architectural structures observed within a given set of exemplars. Here, the architectural model represents *choices* on the part of the domain modelers; but not choices as specific as a single architecture.

7.3.3.3.6.2 Architectural Strategies in ODM

There is an extensive emerging body of research on software architectures, including formalisms, representations, and methods [???]. The intent in defining ODM has been to avoid needless duplication of such work. ODM should be compatible with different architectural approaches; in particular, it allows for selecting appropriate modeling formalisms for each model type.

In addition, ODM involves a process whereby empirical data about architectural approaches specific to the domain of focus can be studied, compared and modeled. This facilitates the adoption and discovery of domain-specific architectural approaches where appropriate, in addition to generally acknowledged architectural paradigms (e.g. the client-server architecture model). This empirical study of architectural solutions within the domain provides the groundwork for systematic discovery of new or previously undocumented architectural approaches; these may be reused in turn within other domains. In this respect, ODM complements the approach elaborated in [Gar93] of discovering a repertoire of architectural idioms that can be applied in a variety of engineering contexts.

Despite this intended "architecture neutrality", there is a set of architectural strategies that appear to be particularly compatible with the ODM modeling approach. The following paragraphs outline a few of these high-level architectural concepts and strategies, connecting each architectural concept to other aspects of the ODM approach.

7.3.3.3.6.2.1 Plug and Play Architectures

Since ODM emphasizes the discovery of "lateral" domains, sub-system-level domains that can be reused in a variety of vertical contexts, a strong emphasis is placed on designing the overall interface to the domain functionality in such a way that the domain system (or subsystem, etc.) is well-encapsulated and easily integrated with other software resources. This requires that sub-domains within the domain are cleanly separated, allowing different implementations to be configured, and that the domain itself can function in a similar way as a sub-system within various other contexts. This architectural approach (sometimes

denoted "plug-and-play") may be quite different from current implementations from legacy systems.

7.3.3.3.6.2.2 Layered Architectures

Closely related to the plug-and-play architectural paradigm described above is the *layered architecture* appproach (described in [Gar93]). Here an inner kernel with minimum or core functionality is surrounded by successive layers, each of which provides enhanced functionality in some respect. The layered architectural principle can be used in a number of ways within the ODM approach, as a response to several distinct kinds of requirements for variability:

Lean vs. full system versions: Often, different contexts of use require different degrees of functionality. Full versions cannot always function as lean version, but few design methods take this notion of restricted functionality as a feature sufficiently into account. The ODM method is intended to help modelers see the potential marketplace and constraints on these different variant versions of assets and architectures. Typically, a lean system version would include assets that in turn support relatively lean subsets of features within the relevant feature models by which they are characterized. Having well-defined specialization within the taxonomy for individual features can help structure similar relations among assets, and by implication among architectural frameworks configuring those assets.

Cascaded versions: Often, layers of functionality can be associated with different contexts of use in a Create - Utilize relationship (from a CFRP standpoint). For example, a value-added reseller typically adds a new layer of functionality onto an existing product, marketing it to a further end-customer. The CFRP provides a framework for modeling such*cascaded* producer-consumer relations, which in some domains can form long chains. This is not a simple special case of the "lean vs. full" distinction. In some cases, the role of the value-added reseller may be to remove extraneous functionality from a base system providing a superset of functionality. This filtering or masking function adds value, in effect, by removing functionality. From the standpoint of pure inheritance and specialization relations, the cascaded version will therefore not always be a fuller version in all respects. A domain architecture may be designed with these relationships anticipated in advance to a degree unusual in ordinary system engineering. These relationships could include tailored, specialized, or instantiated versions.

Variant robustness: One axis of variation for most assets is the degree of error-checking desired in different contexts. For one application context, an extremely robust component may be desired, where the component checks for any possible erroneous use at run-time, returns sensible messages, and does not abort. In another context, where maximum performance is desired, a comparatively more "fragile" version of the component may be desired, that does minimal error-checking. This two extremes with respect to error-checking may be defined as two layers in a set of layered variants for a given asset. Each outer layer would perform some additional context-checking, then call on the next level in.

The same principle can be applied at the larger architectural level for the entire domain system. Each architectural layer can be associated with a set of supported features; outer layers can support supersets of the features supported by inner layers. While this

architectural style has a number of potential drawbacks, it is a natural option to consider in the context of design for reuse.

7.3.3.3.6.2.3 Specialization Layering in Architectures

With regard to the layering strategy outlined above, separate layers will often represent, not structural decompositions of a given system, but taxonomic partitioning into "lean" vs. "rich" functional versions of a system. Such alternative layers might rarely be active in the same application context. Specialization and generalization relations discovered in the course of developing the Domain Interconnection Model during Domain Scoping may significantly impact the layered architectural approaches in Prescriptive Modeling. This may be a way of accommodating variation within distributed, as opposed to encapsulated functionality within a system.

7.3.3.3.6.2.4 Separately Selectable Architectures

Separate selectability refers to accommodation of variation by encapsulated sub-components of an architectural framework that can be included or deleted from system instances relatively independently. When this feature applies to the framework for the domain model as a whole we can refer to each component as a separately selectable feature or component of the domain architectural model. There may be complex constraints on the independent selectability of the sub-components; these constraints would form part of the architectural model.

Separate selectability makes the architectural problem inherently more complex for reuse than in the design of a single system. Consider an architectural model that includes some set of components. In designing a single system, each component function could be designed to be modular and separately selectable on an individual basis although this would be an unusual design strategy for a typical application. For example, few compilers have been architected so that the lexical analyzer, the parser, or the code optimizer is easily accessible as a separate utility. For the case of reuse, however, it may be the case that no one way of partitioning the system into components will meet the needs of the multiple potential contexts of use. The same component may therefore appear as an element of several overlapping separately selectable subsystems obtainable from the overall architecture. In the most complex case, this could lead to a combinatorial explosion of architectural variants for a given component topology. Modelers will therefore typically select a small subset of these combinations to support as separately selectable.

7.3.3.3.6.2.5 Generative Architectures

Finally, it is worth noting that, in the context of reuse, architectural solutions are possible that span multiple contexts, e.g., requiring activity on the part of both the developer and end-user. One example of such an approach would be a *generative architecture*, one in which invocation of a code or application generator is an integral part of the overall system architecture. Such an architecture could not be represented entirely as a static topology of interconnected components.

This architectural strategy may seem to conflict with the aim stated above: namely, that the prescriptive architecture phase should defer asset implementation choices as much as possible, resulting in architectures that will stay relatively stable across technology evolution at the asset level. In the context of the prescriptive architecture for the domain,

however, a generative strategy for linking sub-components may still allow this flexibility of implementation strategies within the scope of each asset ensemble that forms an element of the architecture.

7.3.4 Refine Prescriptive Models (Learn)

This section described the Refine Prescriptive Models sub-phase of the ODM Prescriptive Modeling phase. This sub-phase is analogous to the CFRP Reuse Learning process family within the Reuse Management idiom. It is also analogous to the Refine Descriptive Model sub-phase of Descriptive Modeling. In the latter sub-phase, innovative model transformations were applied before crossing the boundary into Prescriptive Modeling. Here, in addition to integration and validation activities, alternative implementation techniques for the assets in the asset base can be explored.

The processes within this sub-phase are described in a single section. This format is inconsistent with the document structure for other sub-phases of descriptive and prescriptive modeling. It reflects the amount of relevant detail to be discussed, given the current state of maturity of ODM, for these latter steps in prescriptive modeling. It conflicts, to some extent, with the implications of the ODM process tree, which in this regard may err on the side of symmetry and compliance with/specialization of the CFRP process architecture. Further project experience may result in significant expansion of the process descriptions at these levels, or on the other hand may justify re-organization and re-balancing of the ODM Process Tree.

7.3.4.1 Purpose

The purpose of the Refine Prescriptive Models sub-phase is to integrate and validate the prescriptive models created so far, and to explore alternative implementation strategies for the asset base.

7.3.4.1.1 Inputs

Inputs to the Refine Prescriptive Models process may include the following:

Prescriptive Feature Models

Prescriptive Architectural Model

Asset Base Stakeholder Model: used for external validation.

7.3.4.1.2 Workproducts

The primary output of the Prescriptive Model Refinement stage are refined and modified versions of the Prescriptive Feature Models and Prescriptive Domain Architecture.

Common Domain Implementation Model

7.3.4.1.3 Process

The following paragraphs discuss the main activities in this process.

7.3.4.1.3.1 Integrate Prescriptive Models

Prescriptive model evolution has proceeded in a parallel top-down/bottom-up development process. While there are justifications for this approach, it is always necessary after such a development process to allow significant effort for integration of the models produced. In this case, the models include specifications of particular feature variants to be supported, asset ensemble specifications which will implement the feature variants as allocated, and architectural variants derivable from a common architectural model. In the initial step of prescriptive model refinement, these various models are integrated and checked for internal and cross-consistency.

7.3.4.1.3.2 Integrate Architecture with Asset Ensemble Specifications

Once the overall architecture has been established, the asset ensemble specifications can be integrated into this architectur al model. This may involve deriving abstract relationships between the various ensembles, based on the known semantic relationships and constraints among the features and feature sets allocated to the various ensembles.

For example, suppose that one asset ensemble represents a family of variant algorithms used within the domain of focus, while another ensemble represents variants of a key data structure used in the algorithm calculation. Each of these ensembles might eventually by implemented by a family of components, a generative tool, or some mix of techniques. There are constraints between the two ensembles, and some traceability must be preserved from the feature models to the architecture and asset ensemble specifications to ensure that these constraints can still be deduced. That is, for any instantiation of the algorithm asset(s) there may be restrictions on what variants of the data structure asset(s) can be instantiated as part of the same system. Since the asset base architecture defined at this stage links only the asset ensemble specifications as a whole, it can record only the general fact of this dependency between the two asset ensembles. Traceability back to the detailed feature model will also be required.

Integration of the asset ensemble specifications with the architecture can also involve an initial prototyping of a potential asset utilizer's interface to the asset base. The complete implementation of this interface is the responsibility, in CFRP terms, of the Asset Manager, and thus beyond the scope of ODM. However, a prototype of the interface at this point can provide some value. It can serve as an integration vehicle for modelers, domain experts and potential customers to browse and explore the architecture and the assets defined within the architecture. Since the customer contexts have been characterized to some extent in the Planning sub-phase of prescriptive modeling, it is also appropriate to allow for some review on the part of these potential customers in the integration and validation step for the prescriptive phase as a whole.

7.3.4.1.3.3 Validate Prescriptive Models

The integrated prescriptive models are validated, first for internal and cross-consistency (as part of integration) and then with respect to various external criteria. These will include, first and foremost, feedback and review from domain experts and potential asset utilizers. Since these are the prospective customers, identified explicitly, it makes good sense to provide them with the specifications for assets/asset ensembles or overall system configurations derivable from variant architectures. Their reactions will serve as early

validation as to whether the feature combinations and system variants supported are actually deemed to be of use by application engineers.

Where possible, in this validation step, modelers should endeavor to suppress any details about how the assets will be implemented that might influence a domain expert's or application engineer's assumptions about the usefulness of a given variant. On the other hand, modelers should note with interest any constraints or caveats potential utilizers mention about the form, structure, or other characteristics with which assets fulfilling these feature variants would need to comply. These will become valuable inputs to subsequent steps.

7.3.4.1.3.4 Explore Alternative Implementations

In this process, alternative implementation strategies can be considered. The primary input to this process are the asset ensemble specifications, in particular the set of features allocated to each potential ensemble, that serve in effect as inherited requirements that must be fulfilled by the ensemble's design. The output of the process are candidate or draft asset specifications, or high-level descriptions of strategies to be considered. These candidate asset implementation strategies are assessed and finalized in the Planning stage of Asset Implementation.

A full discussion of implementation strategies for reusable design is beyond the scope of this technical report. However, the following three general issues need to be considered:

7.3.4.1.4 Sequencing

The rationale for performing integration, validation and alternatives exploration in the sequence documented here is similar to the rationale for refinement processes in the descriptive phase of ODM.

7.4 Asset Implementation Planning

In Asset Implementation, alternative strategies for implementing the prescribed feature profiles are explored and implementation strategies committed. Key issues to be determined in this exploration stage include trade-off between component-based, generator-based, or hybrid implementation strategies for particular asset ensemble specifications. In addition, possible leveraged development of the new assets can be considered, either from assets drawn from existing asset bases, or artifacts modeled during the descriptive modeling phase. The latter will typically require reengineering to conform to the asset specifications produced by prescriptive modeling In addition, tradeoffs in the structure of the asset base must be considered between strong coupling (with internal reuse) vs. loose coupling (with relative autonomy) of assets within the asset base.

Based on the strategy determined, an overall implementation plan for the asset base is produced, making optimum opportunistic use of generative capabilities, phased or evolutionary development, and bootstrapped use of early components later in the development cycle. A test and validation plan is produced which also takes advantage of the semantic relationships between asset variants within the asset base. The overall process model for Asset Implementation (where ODM addresses only the Planning subtree) is depicted in Figure P.

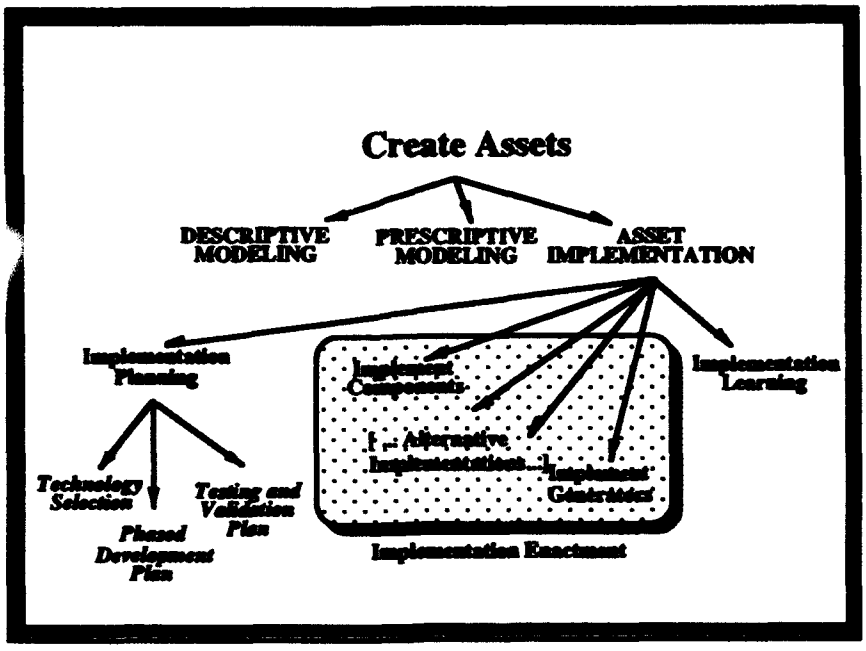


Figure P: Asset Implementation Process Tree

The detailed processes within this phase are discussed below.

7.4.1 Select Asset Implementation Technologies

This process is analogous, in CFRP terms, to the Infrastructure Planning process category within the Reuse Planning family of the Reuse Management idiom. Most of the issues to be considered in this process have been discussed in Section 7.3.4, Prescriptive Model Refinement (Section 7.3.4.1.3.4, Explore Alternative Implementationa).

7.4.1.1 Purpose

The purpose of this process is to make the final selection of implementation strategies for the assets in the asset base, to consider the interaction of these technology choices within the architecture and the asset base as a whole, and to update the architecture model and asset specifications to reflect these decisions.

7.4.1.2 Inputs

Inputs to this process may include the following:

Asset Ensemble Specifications

Prescriptive Domain Architecture

Alternative Implementation Strategies for each Asset Ensemble Specification

7.4.1.3 Workproducts

Asset Specifications: suitable as input to Asset Implementors (component and/or generator developers).

Common Domain Implementation Model.i.Models - ODM: Common Domain Implementation Model;: contains annotations added to the Prescriptive Architecture Model indicating additional constraints and semantics imposed by the technology choices on the architecture.

7.4.1.4 Process

The implementation approach is selected for each asset ensemble within the asset base. The definition of the asset base as an entire collection allows interaction issues to be addressed as necessary along with the local (i.e., asset ensemble-specific) criteria for technology selection.

As each asset ensemble is decomposed into individual assets according to the selected implementation strategy, the asset ensemble specification can be validated for consistency with the individual asset specifications. The detailed plan for how the assets will be implemented is deferred to the following process, Plan Phased Asset Development (See Section 7.4.2). New constraints on the architecture raised by implementation choices are annotated in the Common Domain Implementation Model.

7.4.1.5 Sequencing

In practice, it may not be possible to do architecture development and technology selection in strict succession. Technology choices will require tuning and adjustment of the architecture; architectural choices may impose significant constraints on the technology to be selected. The objective in the process model as described is simply to encourage the definition of a relatively technology-independent architectural layer for the asset base.

There are many sequencing options for how the technology selection process is performed for the multiple asset ensembles, ranging from breadth-first selection, possibly employing parallel teams, to iterative selection, checking and validating each new set of assets added to the asset base in turn.

7.4.1.6 Guidelines

A primary design issue to be considered in this phase is the potential sharing of components among assets. Greater degrees of sharing will lead to a more strongly coupled asset base, which may minimize development effort and consolidate the workproducts produced. However, this strong coupling may work against the goal of separate selectability of subsystems and subsets of assets, as defined by the architectural variants committed to be supported. One possible mitigating strategy to consider in this tradeoff is the use of generative techniques for "smart" configuration management, making the inclusion of required hidden assets relatively transparent to asset utilizers. This design tradeoff is, of course, present in single-system development as well; the issues are merely intensified in the reuse context.

7.4.2 Plan Phased Development

7.4.2.1 Purpose

The purpose of the Phased Implementation Plan (PIP) is to determine the optimum strategy for incrementally implementing the various assets and separately selectable subsystems called out in the Prescriptive Domain Model (PDM). With the Domain Technology Selection phase, basic choices of where components vs. generators will be used have been made. However, there still may be a variety of strategies to follow in implementing the components and generators. The PIP addresses the following goals:

- allows the use of interim products from development on the project itself;
- preserves certain interim versions, if necessary, under separate configuration management so that the version is always obtainable from the asset base and reuse infrastructure;
- helps to select the best strategy for transitioning results of reuse engineering into asset utilization groups that may be at varying points within the application engineering project life cycle.

7.4.2.2 Inputs

Prescriptive Domain Architecture

Asset Ensemble Specifications

Asset Implementation/Technology Selection

Asset Specifications

Prototype Asset Base Interface (if developed)

7.4.2.3 Workproducts

Phased Development Plan

7.4.2.4 Process

Outside-In vs. Inside-Out Implementations. The central design issue of the Phased Implementation Plan lies in the fact that "additive" features (in the sense of the domain model) do not necessarily map to simple sequences of versions to be implemented. Rather than implement a system version with little functionality and gradually add versions that enhance the functionality, it can sometimes be more advantageous to implement the "richer" system version first, then to suppress or strip away the excess functionality to arrive at the "leaner" system versions.

This can be thought of as the "inside-out" vs. "outside-in" dilemma in design for reuse. The dichotomy surfaces at many different points within the ODM methodology. Should "richer" system models be implemented by extending simple initial "threads"? Should "leaner" system models be implemented by masking unnecessary functions from a

superset? These design issues, curiously enough, reappear within inheritance-based modeling heuristics themselves.

Implementation Layers. The PIP may specify additional layers to the architecture than those intended for separate and independent access. Some of these additional layers may serve as "throwaway layers", interim representations used as scaffolding upon which desired persistent architectural layers are constructed. Other layers may be used as "envelopes" to facilitate incremental phasing of new assets and subsystems into existing systems.

Cross-Life Cycle Asset Reuse. In developing the PIP, consideration should be given to interim workproducts, such as domain-specific development tools, that might be used in later phases of asset creation.

Project-specific tools are developed all the time in conventional software engineering; yet, as with many of the practices in reuse engineering, few formal methodologies (with the exception, perhaps of SOFTWARE TOOLS [Kernighan and Plauger???]) provide models for planning development and evolution of such tools as an integral part of systems analysis.

In reuse engineering, since we are designing the "delivery system" for software assets as well as the assets themselves, we are naturally considering development tools as well as assets to be included in the final systems. It makes sense to consider use of these tools for the implementation of the domain assets themselves.

Because ODM considers application development processes as well as artifacts, and considers artifacts and processes across the software life cycle, the method helps in strategic planning to build certain capabilities early in the project to be used later for development activity, as well as becoming part of the final system. This "bootstrapping" approach encourages ODM modelers to follow the same method they espouse (walking the talk), and, on a more practical note, to validate tools developed for the project through direct and immediate usage.

7.4.2.5 Sequencing

Development of the PIP logically follows identification of priorities for separate packaging of domain functionality. To decide how to implement and maintain multiple system versions requires some notion of which versions will be needed, how to minimize required customization, etc.

The PIP also follows selection of asset implementation technology. Implementation phasing strategy might differ greatly, and the PIP take on very different meanings, in component- vs. generator-based reuse engineering. In components-based engineering, the PIP helps sequence the various "builds" or versions of domain assets specified in the architecture. In generator-based engineering, the PIP might detail plans for successively introducing greater levels of semantic completeness in the generator languages as well as plans for using generators and other tools in later stages of the asset creation life cycle.

In general, a separate phasing strategy may be required for each existing or anticipated system into which domain assets are incorporated. The overall plan will reflect opportunities and conflicts stemming from the independent life cycles that each "utilizer" project undergoes. For example, one project might be in an early requirements definition and negotiation phase, where initial versions of domain assets could be used in a

prototyping capacity. A different project may be almost through with detailed design, and therefore only be able to make use of thoroughly tested components that offer a close match to the functional profiles assumed by the current architecture. Each of these life cycle entry points presents different challenges to the phased implementation plan. These issues cannot be deferred until completion of asset creation, however. Such a strategy could wind up with a collection of assets that are not used, despite their technical merits.

7.4.3 Plan Asset Testing and Validation

7.4.3.1 Purpose

Technology Selection, Phased Implementation Planning and Test and Validation Planning can be considered as a specialization of Reuse Planning for the Asset Implementation phase of Asset Creation. The final step in this planning phase is development of a Testing and Validation Plan (TVP) for subsequent Component and Generator Development tasks. Each modeling phase has a separate validation step, as described in the Reuse Learning section below. Here, the implication is that complete testing and validation of a domain architecture occurs in the context of later asset development; hence there are some unique aspects to this validation stage.

This section details issues to be considered in development of a Testing and Validation Plan (TVP) for the domain. The following paragraphs will outline aspects of architecture and asset testing and validation unique (to some extent) to reuse engineering, or that might differ in significant ways from conventional systems engineering approaches. As with many of the techniques discussed in the context of ODM, these testing approaches are not necessarily unique to reuse, and could be applied usefully within the engineering of a single system. However, they by and large represent a departure from conventional engineering practice for individual systems; there is also often much stronger incentive to use these methods in the context of domain analysis.

7.4.3.2 Inputs

Inputs to this process may include the following:

Domain Prescriptive and Architectural Models

Asset Ensemble Specifications (excluding technology selection)

Asset Specifications (including technology selection)

Phased Implementation Plan

7.4.3.3 Workproducts

Workproducts produced by this process may include the following:

Asset Test and Validation Plan

7.4.3.4 Process

7.4.3.4.1 Incremental Testing

The Phased Implementation Plan (PIP) described earlier is consistent with an iterative refinement method of implementation. This naturally suggests use of an incremental approach to testing that derives maximum benefit from the various architectural strategies described above: layered architectures, inheritance-based modeling of system variants, and separately selectable subsystems, among other techniques.

7.4.3.4.2 Cross-Life Cycle Asset Testing

Conventional software testing practices are oriented towards implementations (i.e., code). Asset bases resulting from domain analysis and modeling, architecture, component and generator development (i.e., the Asset Creation process family) will, in general, contain artifacts from across the software life cycle engineered for reuse. The Test and Validation Plan (TVP) should address how the various categories of reusable assets will be tested. In particular, the notion of testing and validation must be adapted to non-code artifacts that have been engineered for reuse. For example, if design schemas are to be included in the asset base, some means of validating the design schemas with regard to their intended scope of applicability must be specified. The design component validation must ensure the design complies with relevant system constraints on form and structure.

In addition, each potential asset specified in the PIP may make use of a spectrum of reuse engineering techniques, from component design to generative reuse, table-driven support for variability, macro expansion or hand-tailored templates. Each of these reuse implementation techniques will require different techniques for testing and validation. Each asset potentially requires testing and validation in several different senses.

7.4.3.4.3 Testing for Conventional Use

Assets must be tested in a manner consistent with their conventional use within a domain application. A reusable code component must be tested in analogous way, but more rigorously, than a component written for a single point of use. The component may need to be tested against several parallel sets of requirements, rather than one set of requirements. The component may need to be tested more exhaustively, to satisfy a higher level of trust. It may need to be boundary-tested more thoroughly, so that its behavior when used in unanticipated contexts can be adequately predicted. It may require system benchmarking tests at lower levels than would normally be required during system test, so that a library of variant components can be adequately characterized with regard to performance trade-offs.

7.4.3.4.4 Testing In Relation to Model

Reusable assets must also be tested, or qualified, with respect to their placement or classification within the domain model that serves, in effect, as their delivery system" An inheritance-based modeling formalism allows relatively strong assumptions about the behavior of a given asset based on its position within the network. Each category within the model can be associated with a set of possible asset implementations satisfying a particular set of requirements or conditions. Specializations within the network must satisfy the same

set of requirements, plus possible additional constraints. Thus much of the testing protocol can be reused in multiple places within the model.

A component that functions appropriately in a single context of use may prove unacceptable when classified within the domain model because of implicit contextual assumptions made in its implementation that violate conditions assumed by the model. Conversely, a version of a component that exhibits non-robust behavior may have a legitimate place within the domain model. (See Section 7.2.3.4.4.6, Error Semantics Modeling .)

7.4.3.4.5 Testing and Validation of Domain Architecture

Earlier it was mentioned that it is difficult to test the domain architecture out of the context of the assets that it supports. Nevertheless, there will probably emerge guidelines for evaluating domain architectures "in their own terms" (i.e., model topology) over time. Such criteria might include: completeness of the architecture with respect to the targeted solution space, as represented by the subset of domain exemplar systems the architecture is committed to cover; consistency of the architecture models (both with themselves, and in relation to other models of the domain); and parsimony (i.e., do models allow specification of assets not intended for support?)

7.4.3.4.6 Use of Inheritance Relations

Because different versions or system configurations are characterized explicitly in terms of addition of features or relaxation or tightening of semantic restrictions, the test plan for the asset base can reflect qualitative criteria for evaluating each more specialized version's compliance with new and maintained compliance with old requirements. Whereas typical unit testing, followed by system integration testing, organizes the succession of test plans in relation to a **structural** model of the end system, use of the domain architecture can lead to testing of increasingly "specialized" versions of assets.

7.4.3.4.7 Prototyping

Simulation and prototype versions of given functionality can be defined within the modeling framework as well. For example, a functional prototype of a component might be a "generalization" that relaxes all performance constraints on the component. A "mockup" version of a component might fulfill all user interface requirements with no underlying functionality implemented. Module stubs could be introduced as extreme examples of "subsetted" requirements of this kind.

7.4.3.4.8 Integration With Asset Qualification

Unlike conventional software testing, the test plans and test data produced for the assets to be engineered can themselves become "assets" within the asset base to be created, integrated into an asset qualification function for the reuse system. These qualification functions could be used by Asset Management or Asset Utilization processes in adding new assets or retrieving and evaluating existing assets. The test plans, test cases and test results can be stored explicitly in the asset base, where they can provide auxiliary examples, and help to document the intended behavior of the assets. (Of course, theoretically, this does imply that these "testing assets", like all other assets, should be tested before inclusion in the asset base...)

7.4.3.4.9 Bootstrapping

"Bootstrapped" tools can be tested and validated by direct use on the project itself, and thus warrant different treatment in the TVP. For example, the testing strategy for a generator will be impacted if the tool is to be implemented early on, then used to generate test scripts for later components. In such a circumstance, the correctness of the component in question will determine how effective a testing tool it becomes; hence rapid completion and validation of tools to be used for "bootstrapping" may be a high priority in the project plan.

7.4.3.5 Sequencing

Development of the PDM can be viewed as the design phase preceding component- and/or generator-based implementation of reusable assets. Defining the TVP after development of the architecture, but before asset development, is therefore analogous to a clean-room design approach where tests are defined directly from requirements. By specifying the testing strategy and even test cases as part of the architecture, the intended semantics of the assets to be engineered are made as explicit as possible. This is not dissimilar to creating a test plan directly from designs in non-reuse-based software development. There are a number of closely interwoven steps in developing the prescriptive domain models, as outlined in the sections above. Test plans at varying degrees of abstraction could be specified at checkpoints throughout this series of steps.

Development of the TVP would usually follow the Phased Implementation Plan (PIP). However, there may be some value in splitting TVP development into black-box and white-box testing strategies. Black-box test plans for variants selected as part of the "delivery space" for the domain architecture could, in theory, be derived before completion of the PIP. Such test plans would need to be formulated independently of assumptions as to whether variants would be generated from a single source file or implemented and maintained as separate modules.

If prepared in earlier stages of domain architecture development, (e.g., before Domain Technology Selection) test plans would need to be even more abstract. For example, if a given range of functionality can be implemented as either a generator or a set of static components, the test data specified for particular system versions might be tested by making run-time calls to a component, or by first submitting a specification to the generator and producing a component optimized to the test data requirements. These tests could therefore be defined on the basis of the asset ensemble specifications.

White-box test plans, on the other hand, would be derived from the PIP, once the sequence and dependencies in implementation were known. These plans would include regression test strategies corresponding to each separate release point in the PIP. Since not every testing point in the PIP represents a version to be maintained under permanent configuration management, not all white box test plans need to be fully "productized" as test assets.

7.4.3.6 Example

OUTLINE DOMAIN EXAMPLE: Suppose that several variant systems had been identified as desired "separately selectable" subsystems for the Outline domain. One variant enforces outline-like constraints on the movement of headings within an outline structure, while another treats the heading movement as a plain and unconstrained text operation. A third

variant accepts a run-time "Preferences" switch which can enable or disable this form of semantic constraint.

Using the descriptive feature model, it should be possible to determine a subset of scenarios where the behavior of the first two variants of the system would differ. Note that these variants are quite different than two modules of a single system that might be unit-tested in a conventional software engineering context. In actuality, only one variant would be likely to be used in any one system instantiation. For the same reason, there is no intrinsic way of defining one version as "more functional" than the other. The relationship between the two variants could be modeled in various ways, and the two variants could also be implemented by a variety of different strategies.

7.4.4 Further Asset Implementation

The ODM method does not address the actual construction of the assets (components and/or generators). By correctly following the steps described above, through the Planning phase of Asset Implementation, asset specifications should be produced that reduce the problem of implementing reusable assets to a tractable design problem. Here, the software development methodology used within the organization can be applied (with the proviso that this application engineering methodology should itself be adapted to utilize where possible, existing workproducts).

Specific techniques required for engineering generator-based assets (or software in general) are outside the current scope of ODM. However, the method is designed to create specifications that allow for consideration of both constructive and generative solutions, and later migration to alternative implementation strategies with minimal disruption to the overall structure of the asset base.

7.5 Reuse Learning

The CFRP calls out an explicit Reuse Learning process family within the Reuse Management idiom to emphasize the importance of systemic learning as a means of maintaining dynamic growth and development of an organization's reuse capability. Ongoing learning is a fundamental aspect of reuse. Without infrastructure and management support for systematic learning, domain-specific reuse could be reduced to the scenario of a fixed domain model, where developers routinely use a common set of reusable components, with no incentive to discover components at higher levels of abstraction or new, useful ways of scoping and defining domains.

In keeping with this emphasis on learning, the ODM method anticipates that domain models themselves will evolve over time, in addition to the assets created and managed with these models. Changes can take place with respect to several different "timelines":

- over the course of the initial domain modeling process;
- over the lifetime of usage of the reusable resources structured with the model;
- over the course of an individual worker's growing familiarity with the domain, reuse tools and domain modeling methods;

- in response to evolution of reuse methods and the supporting infrastructure of reuse tools and environments;
- and with the evolving level of knowledge about the domain within the organization and the technical community.

Since many of these learning processes have been described in more detail within the relevant process descriptions above, the remainder of this section will describe integration, validation and evaluation, innovation exploration and other domain learning aspects at a high level only. The emphasis will be on describing how general CFRP principles have been specialized to the domain modeling context in the ODM method.

7.5.1 Integration

Model integration activities are essential, pervasive parts of the ODM process model. Team modeling has been cited as a core principle and discipline in ODM. In the ODM approach, model development is typically partitioned into reasonably separate tasks, which can be performed iteratively, in parallel or even in round-robin fashion, with circulation of personnel between different modeling and data-gathering tasks. This separate modeling strategy is necessary, even when the models produced are conceptually linked together and could in principle be thought of as one large model. This places a heavy burden on integration activities, which must ensure the consistency of separately developed models, and the traceability of elements across various models.

Integration activities can be interleaved with model development at various levels of granularity, ranging from activities performed by an individual modeler to integration through project-wide review meetings. As the support technology for domain modeling matures, many integration tasks will become better supported by automated tools. In the near term, team modeling skills will be needed for successful application of ODM.

7.5.2 Process Observation

Process observation skills are also core to the ODM method. They are a key element of what has been termed *ethnographic dialogue*, the art of interviewing, observing and participating in the work of domain stakeholders in order to elicit and capture informal knowledge and rationale about the work processes and workproducts particular to the domain.

While the ODM method emphasizes the reengineering of artifacts into assets to facilitate managed, systematic reuse, it could be claimed that the single most important factor in making an artifact from a single-system context of use reusable is capturing the process history and design rationale and other contextual information surrounding the artifact. With this information, a system developer has a good chance of making appropriate use of the artifact, if only as an example or template to work from.

Process observation is also a core discipline in order to identify opportunities for generative reuse, and reuse of domain-specific development processes as well as products. In the ODM approach, domain modeling is viewed as much a modeling of the application engineer's environment as it is a modeling of multiple end-user environments.

Process observation is not only a key element in domain information gathering; it is also an intrinsic element in the structure of ODM workproducts and processes themselves. For

example, the emphasis on separate documentation of candidate lists before selection, selection criteria and rationale, and the documentation of iterative decisions such as the shifts in domain boundaries over the lifetime of the project, are all oriented towards capturing as much of the process involved in domain modeling as possible. This allows for the reuse of various parts of the domain modeling life cycle, and related workproducts, in a variety of subsequent contexts. For example, if a new domain is to be selected in a business area where the ODM planning process was carried out thoroughly, a number of workproducts will have been created that will allow the selection process to proceed much more rapidly, including the Candidate Domain List, the domain characterizations and the Businass Area Lexicon, Stakeholder Model and Domain Interconnection Model. With each subsequent domain modeling effort in a related area, this contextual information base should expand and be refined.

Finally, the capture of process information within the ODM method will facilitate the rapid evolution of the method itself through project experience and lessons learned. Specific modeling process issues to examine may include the following: How much did the concept of the domain change from the initial, informal conception held by modelers: 1) after domain identification, characterization and selection; 2) after domain scoping but before extensive feature modeling; 3) after the completion of the "descriptive" or unconstrained domain model but before architecture and technology selection?

7.5.3 Process Evaluation/Domain Model Validation

The inclusion of a specific objective-setting process within the ODM Planning phase supports a formal process evaluation function as part of the ODM Learning phase, where objectives and success criteria can be measured against project results. The specification of concrete objectives serves other purposes as well; particularly this activity srves to emphasize that domain modeling is always grounded in an organizational context that imposes its own motivating and constraining factors on the domain analysis process.

Overall ODM project objectives are decomposed at every relevant stage into concrete scoping criteria, boundary decisions, rules for inclusion/exclusion, etc. The emphasis in all these activities is quite similar: to transform the vague and ill-specified agenda to "design software to be reusable" into an explicit intended scope of applicability that can serve as the basis for a true design process. Thus, it is an integral concept in the ODM method that **models are not correct in and of themselves.** Models can only be evaluated with respect to specific objectives, a specific potential audience or set of users, and intended usage. These elements form the basis for the model validation processes, which, like other learning-phase activities described above, pervade the entire ODM process tree. The following paragraphs elaborate some major domain validation criteria.

7.5.3.1 Internal Validation

Models can be validated according to certain internal principles, including the following:

1) Completeness: does the model cover all required variability as observed in the exemplar set?

2) Validity: is every concept in the model justified by some precedent within the exemplar set?

3) Consistency: is the model structurally complete? Are all model inter-relationships valid?

4) Expressiveness: is the model capable of expressing variability that would be considered significant by a domain expert/informant?

5) Model balance: is the model structurally balanced? Are there comparable and consistent levels of detail throughout the structure?

6) Minimality (Parsimony): Is there economical use of basic concepts in the model? Are the number of concepts constrained enough to admit a relatively intuitive grasp?

7) Intuitiveness: do model concepts seem natural, or forced/contrived? How good is internalization of the model without recourse to documentation?

7.5.3.2 Process Validation

Was the process of creating the model a valid instantiation of the planned process? Was the actual process documented? Did all joint modelers achieve consensus on the model? This feedback is more likely to impact the modeling process than the models per se.

7.5.3.3 External Validation

The inherently collaborative nature of ODM modeling means that information gathering, modeling and model validation will, in practice, be closely intertwined activities. One reason for breaking out Domain Model Validation as a separate and later phase is to try to consolidate the required interactions with domain experts, whose time (as noted in the Domain Information Gathering process description) will often be at a premium. Just as the domain modeling team should prepare for information-gathering sessions by absorbing as much technical material as possible from written sources, so in validating the domain model it is advantageous to first refine and validate the model using internal resources of the team before drawing in external domain experts for further validation.

Factors such as the degree to which the domain expert feels part of the team, the time pressure involved for the various parties, and the degree of domain knowledge within the team, will temper this strategy. There are complementary risks in over-preparing the materials before going to the experts for feedback and appraisal. Modelers may get too attached to their models after spending greater degrees of effort in polishing them; and experts may be more reluctant to critique models that have the look of finished products. The best strategy, therefore, is to prepare the material as thoroughly as possible, yet leave the presentation somewhat informal and open-ended, so that real feedback can be obtained.

There are many options to consider in obtaining external review, including the following:

1) Use of combined expertise in group review sessions (e.g., experts not accustomed to speak with each other;

2) Use of successively broader review audiences as models become more stable; in particular, try models out on some members of a potential utilizer group **not** part of the program already.

3) Use of key domain informants and/or potential customers for periodic review throughout the modeling life cycle. This helps gain buy-in and ownership from the reviewers and addresses technology transition issues while providing the necessary validation input for the modeling effort.

7.5.4 Innovation Exploration

Innovation exploration has been described in detail in the sections on Domain Innovation Modeling, within the Descriptive Modeling phase, and Exploration of Alternative Technologies, within the Prescriptive Modeling phase (???). In each of these cases an earlier, empirical set of steps act as a kind of "innovation filter", constraining the problem sufficiently so that innovation techniques can be applied in a well-bounded context.

With regard to the learning processes for the ODM method itself, a higher-level Innovation Exploration phase can be considered. Information relevant to this review will be obtained from a variety of workproducts, including: domain boundary decision reports, changes to the domain definition or exemplar set, and the differences between the "prototype lexicon" capturing analysts' starting assumptions about the domain language and the final lexicon obtained from studying artifacts of the exemplar systems.

Some objectives of this analysis might include:

1) to improve workproduct format, the process followed, and/or the guidelines for domain characterization in order to eliminate "oscillation" in domain boundaries (i.e., unstable boundary decisions that move back and forth on key decisions);

2) to accelerate the process of domain definition and scoping;

3) to identify potential new workproducts or processes that could encapsulate common heuristics and decision-making knowledge utilized (e.g., a checklist of review questions for assessing each term of the prototype domain lexicon before beginning extensive information gathering);

4) to contribute data to researchers building theories about variant group decision-making styles and procedures with respect to domain scoping and boundary-setting.

The latter two points would be best addressed after data from several different ODM projects could be studied and compared in parallel.

7.5.5 Enhancement Recommendation

The Enhancement Recommendation process links the Learning phase back to the planning process for the subsequent cycle of the reuse program. Within the ODM context, a number of distinct levels of learning and enhancements can be identified, including recommended changes to and/or evolution of:

1) the domain models themselves;

2) the domain modeling technical infrastructure (tools, starter models, etc.);

3) the ODM workproduct set, process model, or guidelines.

Model evolution may include an incremental migration from informal model semantics (e.g., a checklist, outline, or keyword list) to formal model semantics (a faceted classification scheme, or inheritance-based model), a shift in domain boundaries, transformations in the classification scheme, or gradual inclusion of more heuristic knowledge to guide modeling choices. Different kinds of models may be appropriate at different stages in the various life cycles mentioned above, or more generally for: different domains; different classes of model creators, managers and users; different software engineering cultures of organizations; or different kinds of domain knowledge (e.g., software component descriptions, design rationales, domain-specific error conditions).

Even models of relatively stable scientific domains will reflect the relation of a particular community of knowledge workers to that domain, and can serve as a basis for communication between domain modelers and application engineers; hence model change is potentially a factor in any domain analysis.

8. Comparison with Other Methods

There are now enough domain analysis methods in circulation that comparisons between methods are of increasing importance. The following paragraphs contain some informal observations comparing with ODM with some prominent DA methods. These methods are, like ODM, evolving rapidly with trial use; thus any comparisons are subject to wide errors in accuracy based on the current state of these methods.

One of ODM's express aims was to formalize current DA practice and place it on a more methodical footing. ODM therefore contains steps corresponding to most steps of most DA processes. However, where processes or workproducts incorporated into ODM are analogous to those in other methods, ODM often offers more formal interpretation of what does and doesn't belong in the process or workproduct, and exit criteria for the process. Where a step in another DA process is intentionally excluded from ODM, it is because that step was considered a more specific system modeling step rather than a DA step per se. ???

8.1 ODM and Faceted Classification Approaches

The faceted classification approach to domain analysis introduced by Prieto-Diaz in [ref???] has undergone a major revision which introduced the so-called "sandwich" method of modeling bottom-up (from lexical analysis of domain terms) and top-down (from domain architectures) in parallel. ODM shares DAPM's method's emphasis on combining a bottom-up, domain terminology-based modeling approach with a top-down, architecture-based analysis [Prie91]. However, in DAPM the initial domain lexicon is synthesized directly into a faceted scheme that represents the bottom-up component of the model. In ODM, the lexicon is used as input to a more formal taxonomy, which goes through several transformation step before appearing as a domain features model that can be synthesized in comparatively bottom-up form into sets of variant asset profiles.

Faceted classification can be viewed as a subset of semantic network capabilities, i.e., a "flat" network. ODM views faceted schemes as an intermediate formalism suitable for certain domains or early phases of modeling; inheritance-based modeling formalisms are desirable for the kinds of complex domains in which reuse technology and domain analysis will provide the biggest payoff. Faceted schemes can be represented in RLF, either through somewhat simplified network structures or using AdaTAU "fact bases" (to which they

closely correspond). Representation in AdaTAU, or another rule-based formalism, would aid the heuristically based "closeness matching" that is sometimes possible with this method. One suggestion is that faceted classifications may be a good first simple modeling step that are later refined into inheritance-based schemes as the domain knowledge gets better structured and modelers'' skill increases.

The Prieto-Diaz method (DAPM [ref STARS report???] is quite thoroughly developed in the area of developing the domain lexicon, and many of these steps could be adapted within ODM at this stage. Where DAPM transforms the lexicon into the faceted scheme, ODM would suggest refinement into either/both inheritance-based domain models and/or domain-specific system specification languages.

In the area of architectures, lack of inheritance and specialization in the domain modeling formalism can lead to domain architectures that are basically "generic architectures", capturing the common structural features of systems in the domain but not the variant structures, or forcing the variations to be expressed below an arbitrary level in the architecture.

ODM expresses the common and variant architectural features of the domain in RLF models, reserving the term "architecture", consistent with CFRP terminology, for a later stage of modeling, when design decisions begin to be made about what variants should be prioritized and how they should be configured. This is the step before choosing compositional vs. generative implementation strategies.

8.2 ODM and FODA

FODA uses an inheritance-based formalism (a variant of entity-relationship models) in domain modeling [Kang90]. From the standpoint of formal modeling techniques, FODA's representation for variation in the domain model (mandatory, optional variants and alternative feature relations) form a subset of the expressive power of a full semantic network-based system (a more powerful subset, however, than plain faceted classification).

The AND-OR semantics of the FODA feature model are also, in principle, within the expressive range of a semantic network-based inheritance system. However, it can be argued that the AND-OR representation is much closer to an intuitive language for these constraints than the substructure that would need to be built in an inheritance system to capture the same semantics. How essential the extra power of a semantic network system like the KL-ONE variant supported by the RLF will prove to be for domain modeling will become clear only after more case studies have been completed.

FODA and ODM also share a common notion of "feature modeling" (a term that also appears in other published methods such as [Gomaa???]).

ODM's approach to feature modeling resembles FODA's user-visible requirements-oriented feature modeling approach in a number of respects [Kang90]. The prototype use of ODM on the RLF design introduced the notion of feature analysis at close to the same time as the FODA work. The notion of what constitutes a feature appears to undergoing evolution in both methods. Originally, the easiest way to characterize a feature was in terms of a functional capability that would be visible, or of interest, to an end-user of a system in the domain. It has become clear that the domain model needs to be able to discuss other system aspects as differentiating features, including aspects that may be buried far below the level of

direct user interaction or direct user visibility in terms of operational consequences. The notion of feature therefore has broadened in both methods.

However, ODM has now evolved towards a stricter interpretation of the feature concept, to serve as a construct that integrates several of the descriptive conceptual models. Thus, the ODM method could be said to model features at a comparatively lower level of semantic granularity. It remains an area of further research how much the formality of ODM can be downscaled to projects with very limited resources, without removing key elements that support the integrity of the underlying conceptual process in ODM.

ODM has a richer set of domain models that includes a number of FODA modeling notions (like feature binding times), but also includes models for other artifacts across the life cycle. Some of FODA's models would be classified as system models, according to ODM distinctions between "system model fragments" and domain model artifacts. For example, FODA's dynamic, finite state machine characterizations of system behavior are arguably not domain models in the strict ODM sense. It is questionable how specialization relations really work for these types of models, and it is by no means obvious how to extend or to defer these models to express system differences without making strong implementation assumptions.

8.3 ODM and OO Techniques

Semantic network modeling provides a subset of object-oriented features. The ODM method is based on the hypothesis that some object-oriented features (e.g., code inheritance) are less relevant to domain modeling than to implementation of object-oriented systems. While compatible with object-oriented analysis (OOA) and object-oriented design (OOD) techniques, the ODM method does not presume that these techniques will be used in development of applications in the domain.

9. Future Work

9.1 Near-term

The highest priority for further development of the ODM approach is to continue to validate and refine aspects of the model in practical engineering contexts. Extensive domain modeling will most often be done by domain experts themselves rather than domain analysts per se; reuse specialists will be called on at most for collaboration, training, and advisory review sessions. Thorough validation of the ODM approach will therefore depend on the results of trial usage by early adopter groups willing to risk applying methods still in evolution (with appropriate management support).

Future versions of this technical report and/or related documents should include the following:

- Syntax and semantics for formal specifications or mini-languages for many of the models called out in the approach;
- Tailorable process documentation (e.g., IDEF or SADT-style diagrams) for ODM processes;

- A more complete lexicon of terms with special meaning in the ODM context, normalized to address discrepancies in the current text, and integrated with similar lexicons for CFRP and general reuse and software terminology;
- A complete set of workproduct examples, suitable for a tutorial, preferably drawn from a project of significant size;
- More detailed comparisons of ODM with other DA methods in use, and with methods from related disciplines;
- More practitioners' guidelines, from both a project manager's and a domain engineer's point of view;
- A prototype set of ODM resources, such as starter models for particular ODM workproducts, checklists and validation criteria.

9.2 Mid-term

Important areas for further theoretical and technical work include:

- integrating the ODM approach with specific domain modeling tools such as the RLF toolset, commercial outline processors and/or hypertext systems;
- formalizing the semantics of inheritance for domain models (in particular the problematic area of procedural models);
- relating the organizational perspective of the ODM to overall reuse process and organizational models (such as the STARS CFRP) to provide guidance to planners initiating reuse programs;
- "stress-testing" the ODM process model by applying it to non-standard "software-like" domains such as font data, digital samples, video images and other multimedia artifacts;
- integrating the ODM with current work in organizational change and redesign, organizational learning and workplace ethnography.

9.3 Long-Term

In the near term, ODM models will facilitate reuse across applications within the same domain. In the longer term, ODM's emphasis on tightly scoped, modular domain models and multi-faceted domain interconnection models should prove useful in exploring "inter-domain" reuse, i.e., potential commonality across systems in different domains and business areas. Perhaps the most critical application for inter-domain modeling lies in conversion strategies for the defense software industry. As defense contractors search for ways to leverage existing systems and communities of technical expertise to respond to radically transformed military and political scenarios and new global and commercial markets, domain modeling could aid in:

- consolidating multiple existing systems into commonly supported and maintained bases of engineering artifacts;

- methodically exploring commercial markets for re-targeted military system capabilities;
- re-architecting systems for scalability to state or local levels, or globalization of previously nationally scoped systems;
- identifying potentially reusable sub-systems within existing vertical application areas; and
- helping to shape strategies for retraining and re-targeting of workers with specific technical skills for new application areas.

Current economic and political shifts are creating conditions where the potential benefits of reuse have a better chance of being realized in the next few years. Hopefully, ODM will have a contribution to make as the reuse community addresses these challenges and opportunities.

10. Conclusion

This technical report has presented the basic processes and workproducts of ODM, along with some background in underlying principles that integrate these elements into a coherent method for domain modeling. ODM proposes a new degree of formality for domain analysis, a still-emerging discipline that has heretofore been performed in a somewhat ad hoc manner. ODM brings an organizational perspective that significantly reinterprets the role and potential impact of domain modeling within software engineering organizations. The essential notions of clearly defining the intended scope of applicability for reusable assets, differentiating descriptive from prescriptive modeling, and the explicit inclusion of a systematic approach to innovation within domains are examples of areas in which ODM has made some progress. While the work clearly has much farther to go, it is hoped that this technical report will serve as a useful snapshot of progress to date, and as an aid to prospective modelers towards applying ODM in practice.

A. ODM Guidelines for Domain Identification

An organization's reasons for initiating a domain analysis, choice of domain, and even the scope within which domains are to be identified, may occur very differently in different business scenarios. In some cases the domain of interest will be defined in advance by obvious business constraints. In other cases, the appropriate domain will be selected only after a long and methodical comparison of candidates. Depending on this business context, the domain analyst may face the problem of: comparing multiple domains with the aim of selecting the one with the greatest potential return on reuse investment; analyzing an existing collection of resources with the aim of defining one or more domains discernible within the collection; or evaluating a loosely defined line of business or application area with the aim of scoping one or more tractable domains within its purview, and advising whether or not to proceed with further reuse efforts. Issues relative to several of these possible reuse business scenarios are discussed in the following paragraphs.

A.1. Domain Selection by Management Directive

A prevalent context for domain analysis is a management directive to examine a particular application area. In this situation, the domain of interest is determined by wider business needs. A primary function of the preliminary domain analysis will therefore be to advise a go/no-go decision. Does the company stand to benefit from investing effort in reuse technology within this area? A subsequent question is selection of appropriate technology. Is there a technology or mix of technologies that will produce significant productivity improvement in this area? In general, if domain choice is highly constrained, then choice of technology must leave some room for flexibility.

Some specific situations that might lead to such a management directive are discussed below. These include: transition from a contract- to a product-based development paradigm; a perceived need to put requirements specification on a more rational basis for an application area; consolidation of redundant application groups as a result of corporate mergers, acquisitions, reorganization or other trauma; or a changing competitive environment for an established line of applications.

A.2. Transition from contract- to product-based development

An ideal point in an organization's growth to initiate domain analysis for a specific application area is when a number of contracts have been successfully completed by the organization, and the prospect of recurring business is strong. Depending on the size and complexity of the application area, this is often the point where there is movement within the organization to shift from a contract- to a product-based strategy in the domain. This impulse may stem in part from reaction to problems that, in a sense, grow more critical as the success of the business increases. Contract-driven organizations often attain their initial niche in a market by customer responsiveness—a willingness to customize and tweak systems to accommodate the business practices and policies of different customers. This flexibility on the part of the developers eventually begins to create problems in configuration management, maintenance costs, overall reliability, or simply the growing amount of developers' time. A shift to a product-based approach signifies a new willingness to establish criteria for what the business can and cannot accommodate.

However, developers soon find that the boundaries of this emerging "product" concept cannot be specified by matching to an explicit set of requirements, as were earlier contracts. Furthermore, if built by conventional methods, the resulting product may, in trying to be too many things for too many customers, wind up too little for too few. In these situations, the process of determining the appropriate product scope (or the customizable generic system) is essentially a domain analysis, though it may not be called this or formalized to any great extent. A domain analysis process model should offer guidelines for techniques to use at this stage of the domain life cycle. By offering alternative conceptions of "productization", such as the use of program generation techniques, or separately selectable and scalable configurations, a reuse-based methodology may suggest better ways of covering the range of functionality needed in potential systems.

A.3. Recurrent Specification Problems

For many contract-based software development organizations, the need for a shift in approach may surface first in recurring problems in system specification. Even if it appears likely that programmers will continue to assemble systems for new clients by manual production ("plain dirt programming"), improved methods for validating the completeness, consistency, and economy of new customer specifications, relative to the knowledge of a particular domain, can be a great asset in the way the company does business. This may involve consolidating the applications knowledge of experts who may be scattered throughout the company. In such a scenario, the initial emphasis might be on creating a domain model of system requirements, rather than a model for reusable code per se. Once a domain-specific, semi-formal specification language has been defined, there are often opportunities for automating software production via libraries of components or generative techniques that may not have even been apparent before definition of the language. However, in such situations it is advisable to base specific objectives for the project on the achievement of reuse at the requirements level, since these benefits are more predictable and will be measurable in the shorter term.

A.4. Crises of structural reorganization

Crises of structural reorganization can also provide motivation for a domain analysis effort within an organization. For example, corporate mergers and acquisitions often create situations where overlapping and/or redundant systems or lines of business must be consolidated and reconciled (with one perhaps being phased out). In such situations, a comparative analysis of the capabilities of the various systems, relative to the anticipated needs of the marketplace, is often a critical part of strategic planning. Here, a methodology that deals with an existing "population" of software systems and their features is needed.

A.5. Competitive Analysis

Companies must often analyze their software products or services relative to those of competitors in the marketplace to determine issues such as targeting a specific market niche (high vs. low-end), packaging and bundling of options, and price-performance positioning. This competitive market analysis may take on aspects of domain analysis, depending upon the extent to which the analysis relies on systematic exploration of variant feature profiles of competitor and potential systems.

A.6. Domain Identification By Induction

In some cases, the choice of domain is not fixed by the business context, but the choice of exemplar systems is. An example would be a division with responsibility for a specific set of systems. The choice of domain may be relatively unconstrained; however, the examples must be drawn from this set of systems.

Issues to consider in this scenario include the following:

- Is it possible that the collection of artifacts or systems are so poorly articulated with respect to domain that any attempt at domain identification will be futile? In other words, will there be an adequate representative set for any domain within the collection as given? If the collection has been gathered with some rationale having to do with line of business, then domain identification may work well. However, a collection resulting from ad hoc gathering of software materials (such as a collection of freeware or shareware, or an archive) may not be amenable to an overall domain modeling approach. Here, only historical and accidental factors result in the clustering of the materials for one analysis.

- What classification scheme (formal or informal) is currently used by those accessing the collection? Is there process information resulting from the storage and retrieval method for the collection that might be of value for the domain modeling effort? For example, would statistics on numbers of accesses, numbers of usage, length of time in repository, etc., be available?

- Tailoring: If the scope of the exemplar system collection is definite enough, it would be feasible to do the Enterprise Software Inventory before completing domain selection. In this case, the inventory properly belongs to the Business Area Portfolio, since presumably the same inventory will be utilized by multiple domain modeling efforts.

A.7. Technology-driven Search

A final business context for domain selection involves searching for a domain suitable for a particular new technology to be applied. This scenario will be most likely in an organization that has a substantial commitment to a technology developed in-house, such as an R&D organization, or to a supported commercial product that is already part of the technical infrastructure for the organization.

As an example of opportunistic technology selection: if an application generator system had been developed in a corporate R&D group and needed to be demonstrated in use within the company, a domain modeling project could be defined to produce a domain-specific language to be implemented with the generator. In this scenario, the commitment to the generative approach is a constraint in the context rather than a feature of the ODM method itself. However, the ODM method will provide direct support in defining the requirements for a generative solution within the domain, whereas DA methods based on an implicit components-oriented approach might not produce the information in the optimum way.

A.8. Technology-Driven vs. Domain-Driven Organizations

One major variant in the sequencing of process categories within Reuse Planning lies in the distinction between *technology-driven* vs. *domain-driven* organizations. Both of these profiles differ from what might be termed the *project-driven* organization, that is, an organization under

so much stress and schedule/budget pressure that the decision to invest in reuse (either through infrastructure development or domain analysis efforts) cannot be made. For organizations that do have the corporate will to initiate reuse efforts, at least two contrasting approaches can be identified.

In the technology-driven scenario, the availability or the incentive to develop key reuse technologies takes precedence over the choice of domain, which is in fact influenced in part by the suitability of the domains considered for the technology of choice. This approach would be most typical in research and development organizations (either autonomous firms or R&D divisions within larger companies), academic or educational settings, or commercial vendors of technology relevant for reuse. In tailoring the Reuse Planning family in such a scenario the Infrastructure Planning category would precede Domain Selection. (It would still be advisable to allow the identification and characterization of domains, currently described within the Assessment category, to be as unconstrained by technology assumptions as possible.)

The second approach starts with the needs of application groups who have particular problems in particular lines of business to address. This approach would be more appropriate for commercial or contract-driven software development groups, and in this approach domain selection should probably precede a commitment to a particular reuse technology.

ODM is flexible enough to accommodate a spectrum of different organizational scenarios with respect to this point of variation. It does make clear the need for caution when business constraints have prematurely fixed choice of both technology and domain, since these may not be an appropriate match, and this may not be clear until well into domain analysis. Similarly, it calls for caution when two organizations who may be driven by very different factors have a key interface; for example, the typical kinds of problems of technology transfer that occur between R&D and applications divisions of large software development organizations.

B. ODM Lexicon Specification Language Definition & Example

```
--:: Title:     BNF for Domain-Specific Lexicon (DSL)

--:  This is a sample specification for an ODM MSL (model-
--:  specific language) for the Domain-Specific Lexicon. It is
--:  currently in draft form, and is meant to suggest what
--:  such a language specification might include and how it
--:  could be reflected in an example workproduct based on
--:  the language specification.

Domain-specific-lexicon ::=

Standard-descriptor

lexicon LEXICON_NAME is
      for domain DOMAIN_NAME
    [ includes LEXICON_NAME {, LEXICON_NAME } ]
    [ term types:
      Standard-Descriptor
      TERM_TYPE_NAME { , TERM_TYPE_NAME } ; ]

--:: Rationale:
--:  Special term types need not be defined; if the term
--:  types clause is included, though,at last one term type name
--:  must be provided. If only one term type is provided, it is
--:  the default for the lexicon and any terms without a
--:  specific type assigned take on this type.

terms
term TERM_NAME is
      Standard-descriptor
      [ type TYPE_NAME ; ]

--:: Issue:  should multiple term types be allowed?

      [ acronym: TERM_NAME ; ]
      [ synonyms: TERM_NAME {, TERM_NAME} ; ]

--:: Constraint: a term can't be its own synonym...
      [ paraphrase: [ TERM_NAME  | LIST_WORD ]
                    {, [ TERM_NAME  | LIST_WORD ] } ; ]

--:: Rationale: intent of paraphrase keyword is to allow compound
--:  phrases that restate the intended semantics of the term in
--:  relation to other terms in the lexicon or standard words
--:  from a wordlist (e.g., "of", "by" etc.)
--:: Constraint: a term can't be used in its own paraphrasis ...

      [ sources:  Domain-Info-Ref {, Domain-Info-Ref } ; ]
;
end terms;
end lexicon [ LEXICON_NAME ] ;

--:: Constraint: LEXICON_NAME, if included, must match start...
.
```

```
--:: Title:     OUTLINER_LEXICON

lexicon OUTLINER_LEXICON is
      for domain OUTLINER_DOMAIN ;
      sources:  acta_advantage_exemplar ;
term types
      operation, object, relation;
terms
term sister is
      type relation ;
      synonyms:  sibling ;
;

term mother is
      type relation ;
      synonyms: parent ;
      inverse: daughter;

--:: Issue: couldn't symmetrical inverses be written once in the
--:   spec, then other clause generated via some prettyprinter?
--:   Otherwise, at least requires a consistency check...
--:: Issue: does this language tolerate forward references?

term daughter is
      type relation ;
      synonyms:  child ;
      inverse: mother;
      sources:  acta_advantage_exemplar ;
;

term next is
      type relation ;
      synonyms: subsequent ;
      inverse: prior ;
      sources: acta_advantage_exemplar ;
;

term aunt is
      type relation ;
      paraphrase: next sister of parent;
      sources:  acta_advantage_exemplar ;
;

end terms;
end lexicon OUTLINER_LEXICON;
```

C. References

[Aran91] Arango, G., Prieto-Diaz, R., "Domain Analysis Concepts and Research Directions", in **Domain Analysis and Software Systems Modeling**, IEEE Computer Society Press, 1991.

[Bail92] Bailin, S. C., **KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Rationale**. CTA Incorporated, Rockville Maryland, 1992.

[Bat88]? Batory, D. **Concepts for a Database System Compiler**. TR-88-01, University of Texas, Austin, 1988.

[CAMP87] **Common Ada Missile Packages**. Technical Report, 1987.

[Coad89] Coad, P. **Object-Oriented Analysis**. Object International, Austin,Texas, 1989.

[Col91] Collins, P. "Toward a Reusable Domain Analysis," **Proceedings**, 4th Annual Workshop on Software Reuse, Reston Virginia, November 1991.

[Col92] ibid., "Considering Corporate Culture in Institutionalizing Reuse," **Proceedings**, 5th Annual Workshop on Software Reuse, Palo Alto, CA., October 1992.

[Cre92] Creps, R.E., M. A. Simos, R. Prieto-Diaz, "A Conceptual Framework for Reuse Processes", STARS informal report, September 1992 (available from authors).

[DISA93] **Domain Analysis and Design Process**, Version 1.0, DISA/CIM Software Reuse Program, March 30, 1993.

[Gar93] Garlan, D., Mary Shaw, (School of Computer Science, Carnegie Mellon University). "An Introduction to Software Architecture", **Advances in Software Engineering and Knowledge Engineering**, Vol. I, World Scientific Publishing Company, 1993.

[Kang90] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[MC89] McDowell, R., and K. Cassell. "The RLF Librarian: A Reusability Librarian Based on Cooperating Knowledge-Based Systems", **Proceedings**, 4th RADC 4th Annual Knowledge-Based Software Assistant Conference, September 1989.

[PW91] Perry, D.E., A.L. Wolf, "Software Architecture", unpublished paper, January 1991.

[Pra90] Prahalad, C.K., and G. Hamel, "Strategic Intent", **Harvard Business Review**, May-June 1989.

[Prie91] Prieto-Diaz, R. **Reuse Library Process Model**. Technical Report AD-B157091, IBM CDRL 03041-002, STARS, July 1991.

[Sim86] Simos, M. "Alternative Technologies for Software Reusability", **Proceedings**, 3rd STARS Workshop on Applications Systems and Reusability, Washington, D.C., February 1986.

[Sim87a] ibid. "Gadfly: A Knowledge-Based Software Unit Test Plan Assistant", **Proceedings**, Unisys Software Engineering Symposium, September 1987.

[Sim87b] ibid. "The Domain Life Cycle: Steps Toward a Unified Paradigm for Software Reusability", **Proceedings**, 1987 RMISE Workshop on Software Reuse; reprinted in **IEEE Tutorial on Software Reuse: Emerging Technology**, ed. William Tracz, IEEE Computer Society Press, 1988.

[Sim88] ibid. "The Growing of an *Organon*:: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse", invited presentation, 1988 NISQP Conference on Software Reusability; reprinted in **Domain Analysis and Software Systems Modeling**, ed. R. Prieto-Diaz and G. Arango, IEEE Computer Society Press, Los Alamitos, CA., 1991.

[Sim90a] ibid. **Determining Requirements During Domain Analysis**, Software Productivity Consortium Distinguished Lecturer videotaped presentation, February 1990.

[Sim90b] ibid. **Alternative Approaches to Domain Analysis: Steps Toward a Job Description**, course notes, 1990.

[Sim91a] ibid. "Software Reuse and Organizational Development", **Proceedings**, 1st International Workshop on Software Reuse, July, 1991, Dortmund, Germany.

[Sim91b] ibid. "Navigating Through Soundspace: Modeling the Sound Domain At Real World", **Proceedings**, 5th Annual Workshop on Institutionalizing Software Reuse, Herndon, VA., November, 1991.

[Sim92] ibid. "Towards An Industry-Wide Consensus Reuse Process Model", **Proceedings**, 5th Annual Workshop on Software Reuse, Palo Alto, October 1992.

[SWT89] Solderitsch, J., K. Wallnau, and J. Thalhamer. "Constructing Domain-Specific Ada Reuse Libraries", **Proceedings**, 7th Annual National Conference on Ada Technology, March 1989.

[STARS92a] Software Technology for Adaptable Reliable Systems (STARS). STARS Asset Library Open Architecture Framework, Version 1.0. Informal Technical Report; STARS-TC-04041/001/01, DARPA, March 1992.

[STARS92b] ibid. STARS Reuse Concepts, Volume I: Conceptual Framework for Reuse Processes, Version 2.0. Paramax STARS Technical Report STARS-TC-05159/001/00, DARPA, November 13 1992.

[STARS93] ibid. STARS Conceptual Framework for Reuse Processes, Volume II: Application. Version 1.0. In preparation; available August 1993.

[Unisys88] **Reusability Library Framework AdaKNET/AdaTAU Design Report**, Naval Research Laboratory, System Development Group, Unisys Defense Systems, PAO D4705_cv_880601_1, 1988.

[Wall88] Wallnau, K., J. Solderitsch, M. Simos, R. McDowell, K. Cassell, D. Campbell. "Construction of Knowledge-Based Components in Ada", **Proceedings**, AIDA-88, George Mason University, November 1988.

[Wartik92] Wartik, S., R. Prieto-Diaz. "Criteria for Comparing Domain Analysis Approaches," **Int'l Journal of Software Engineering and Knowledge Engineering**, September 1992.

D. ODM Lexicon

Conventions in the Lexicon:

Bracketed text in the name (e.g., [software] domain) means the bracketed phrase is optional, and is part of the assumed context when the unbracketed phrase is used within discussions of ODM.

Activities can have names as verb phrases (Identify Domains) or noun phrases (Domain Identification) for the purposes of sentence context. They are listed here only once.

In workproduct names, singular means this entity scopes what is dealt with in the model; plural means the model provides comparison of multiple entities of this kind.

Within the text body for a lexicon entry: other terms used in the definition and description that appear elsewhere in the lexicon are*italicized* the first time they are mentioned within that block of text. Sub-terms may be introduced within the text body that are not separately listed in the lexicon, but represent specializations of the concept denoted by the main term; these will be listed in **bold-face** type the first time they occur in the text. Finally, some of theese sub-terms may also be listed elsewhere in the lexicon, for the sake of easy cross-reference; these are listed in ***bold-face italic*** the first time they occur in the text.

D.1. ODM Models

Business Area Domain Interconnection Model

Model of domains of interest within the business area for the reuse program. May be generated at the level of Reuse Planning for the reuse program as a whole, or synthesized from the Domain Interconnection Models generated by one or more Asset Creation/domain modeling projects. Unlike these models, the Business Area Domain Interconnection Model is not focused around a single domain, but rather attempts to identify all domains of interest within the business area.

Business Area Stakeholders Model

Model of the various stakeholder groups within the business area context. Forms the superset of the analogous model for any particular domain within the business area. Like the Business Area Domain Interconnection Model, may be generated during Reuse Planning for the reuse program or synthesized from the Domain Stakeholders Model from one or more Asset Creation/domain modeling projects.

Common Domain Implementation Model

Model that elaborates the [Prescriptive] Domain Architecture after the Technology Selection process has been performed for the various asset specifications within the domain. Accounts for any impact of these technology choices on the architecture. Provides input to the Phased Implementation Planning process within Asset Implementation Planning.

Descriptive Domain Models

The set of all models produced during the Descriptive Domain Modeling phase, including the Feature Binding Times Model, artifact, conceptual and/or contextual models, descriptive feature models, the Descriptive Features Model. When referenced within later processes of the Descriptive Domain Modeling phase, denotes the cumulative set of models produced so far; when referenced outside that context, denotes the entire set of domain models linked to precedents within the Domain Exemplar Set, up to the Interpreted Domain Model.

[Descriptive] [Domain] Feature Model

A model of a domain feature, as defined in relation to terms in the descriptive artifact, conceptual and contextual models. Includes the significant variants for the feature, where each variant is traceable to a precedent in at least one exemplar, either through or independently of the underlying descriptive models.

Domain Architectural Model

An ambiguous term in the ODM context, as it could refer to either a descriptive or prescriptive model. (In context, prescriptive is usually implied.) A model of system or subsystem variants that span the entire range of functionality for the defined domain. Addresses issues such as the separate selectability of major components of the architecture, scalability or layered variants of the domain functionality as a whole, and the interfaces to surrounding system context. May imply a topological schema (i.e., a high-level design) for the domain scope, a principle-based architecture used to generate detailed design decisions, or a configurable and adaptable architecture.

Domain Features Model

The **unified** and **integrated** model of feature variants synthesized from the individual Domain Feature Models during the Domain Model Integration process. This model includes semantic relationships between features, based where possible on implications of the semantics defined on the underlying descriptive artifact, conceptual and contextual models.

Domain Genealogy Model (DGM)

Model characterizing historical interconnections between systems in the domain. The scope of systems addressed in the DGM can vary from those in the Domain Definitional Set, Exemplar Set, Qualified Exemplar Set, or Representative Set, depending on criteria such as the number of potential systems to be considered (i.e., the danger of expending resources on irrelevant systems), the possible value to the organization of a systems inventory decorated with genealogical information as a resource beyond the context of the ODM project, or the existence of clear principles for grouping certain sets of systems as within the scope of interest of the organization.

Domain Information Types Model

Model of the potential types of information sources to be investigated for the domain. Differs from the Domain Information Sources Catalog in that specific sources are not mentioned; can be considered the schema or generic model for that catalog.

Domain Innovation Model
The model produced during the last process in Domain Model Refinement, when the Interpreted Domain Model is transformed using a repertoire of model innovation strategies to discover novel combinations and permutations of existing features and architectural variants. Variant features within this model may have no known precedent within the Domain Exemplar Set.

Domain Interconnection Model (DIM)
Model of the qualified relations between the domain of focus and various related domains. Produced during the Domain Scoping activity of the Context-Setting process of the Descriptive Domain Modeling phase. Used to identify major domain variants in later descriptive modeling, possible innovative feature combinations during Domain Model Innovation, and, in general, to ensure a well-bounded domain focus. Each relationship in this model involves the domain of focus, unlike the Master Domain Interconnection Model which has the scope of the entire organization or business area.

Domain [-Specific] Lexicon
An annotated lexicon of terminology within the domain produced as part of the Domain Information Generation process. ???

Canonic Domain Lexicon
The Domain-Specific Lexicon, with annotations of exemplar precedents, and with a canonic term chosen as the representative to use for synonymous classes of terms drawn from various exemplar artifacts. This provides a normalized syntax to use in other models for the domain.

Precedented Domain Lexicon
The Domain Lexicon, annotated with linkage to exemplar artifacts for every term in the lexicon.

Prototype Domain Lexicon
The lexicon created based on the informal domain knowledge of the modeling team.

Domain Ontologies Model
A model that serves as a plan for what specific models to develop during Descriptive Domain Modeling. The models are loosely related semantically. Each model is named and its *ontology* specified, that is, the types of instances the model is intended to encompass. The model is developed during the Determine Domain Model Coverage activity of the Domain Context-Setting

Domain Prescriptive Architectural Model
Domain Prescriptive Feature Model
Domain Stakeholders Model
A restriction/specialization of the Business Area Stakeholder Model, including all and only stakeholder contexts applicable to exemplars in the domain of focus. The model is developed and/or formalized during the Organization Assessment process of the Reuse Planning phase, and used to describe who's interested in the feature as a differentiator and to whom it isvisible.

Although logically the Domain Stakeholder Model specializes the Business Area Stakeholder Model, in practice this model will often be developed first. Modeling at the more specific level often reveals not only lower-level, but also higher-level details undiscovered during modeling in the higher-level context.

Feature Binding Times Model
Integrated Domain Model
Interpreted Domain Model
Modified Asset Base Stakeholders Model
Organization/Business Area Stakeholders Model
a model, structured in terms of CFRP Asset Creation, Management, and Utilization process families, that details the reuse-specific relations within the program context.

Stakeholder Profile Model

D.2. Other ODM Workproducts

Business Area Information Sources Catalog
Candidate Domain Characterization
Candidate Domains List
Domain Analysis Objectives Statement
Domain Asset Base Intended Scope
Domain Boundary Decision Report
Domain Characterization Criteria
Domain Definition Statement (DDS)
a textual description of inclusion and exclusion rules for systems or subsystems within the domain. Includes features or capabilities considered to be defining characteristics for the domain, purposes of systems in the domain

Domain Definitional Set
includes all candidate systems considered as potential exemplars for the domain. This set may be extracted from the Organization/Business Area Systems Inventory, selecting systems identified as potential exemplars, based on the informal initial domain understanding that is later formalized in the Domain Definition Statement.

Domain Dossier
Domain Dossier Structure
Domain Exemplar Set (DES)
includes a first cut of the Domain Definitional Set into core, member, borderline exemplars and counter-exemplars

Domain Identification Criteria
Domain Information Corpus
Domain Information Sources Catalog
Domain Model Validation Plan
Domain Modeling Formalisms Selection
Domain Representative Exemplar Set
Domain Representative Set (DRS)
Domain Selected Ontologies Set
Domain Selection Criteria List
Domain Selection Report
Domain-Specific Specification Language (DSSL)
Domains Portfolio
Enterprise Software Inventory
Learning Recommendations
Model Boundary Decision Report
ODM Project Handbook
ODM Project Plan
ODM Project Process Model
Organization Profile
characterization of the organization with respect to the set of features in the ODM Organization Typology Model

Organization/Business Area Systems Inventory
Qualified Domain Exemplar Set
integrates the DES with the DIM, to identify, for borderline exemplars and counter-exemplars, what related domains best describe the interface issue raised by the exemplar

Resource Plans
Reusability Analysis Report

D.3. ODM Processes

[TBD]

D.4. ODM Roles

The ODM method distinguishes roles at the individual, team and organizational level. Many of these named roles overlap with roles specified in the CFRP.

asset developer
domain expert
[domain] modeler
A person performing modeling activities in the context of an ODM project. When used without a qualifier, a *modeler* is presumed to be a domain modeler.

domain informant
domain stakeholder
system modeler

A person performing system modeling activities, usually within the context of an *application engineering* project.

ODM Project

A *project* (in the CFRP sense of the term) oriented around domain engineering for a particular selected domain. The project may be part of a larger reuse effort that includes, for example, a system reengineering effort, a corporate reuse planning effort, etc. The **ODM project** refers in this case specifically to that portion of the larger effort committed to performing the activities within the scope of ODM. This boundary may become somewhat arbitrary, particularly since ODM planning and learning stages are expected to merge with broader reuse planning and learning activities, and also because circulation of project members and periodic joint activities are legitimate strategies for managing reuse programs of this kind.

ODM [Project] team

Somstimes termed the **Domain Modeling Team**: the team performing the ODM project as described above. Note that a *domain engineering* team would normally denote a broader scope of activities, including in particular asset implementation. ODM activities lead up to but do not include asset implementation.

D.5. ODM Resources

We will term any standard elements of the ODM documentation set intended to assist in the ODM process as an *ODM Asset*. By calling these workproducts assets, we are asserting that they are under management in some Reuse Engineering context. That context is the context where the ODM Development Team makes these assets available to domain modeling teams in various settings, and receives feedback on the ODM assets that are incorporated into successive refinements of the assets as part of the Asset Management cycle. Since this context is not yet formally established, these documents are referred to at present as ODM **Resources**.

Within the context of a domain modeling project, therefore, the ODM assets will be utilized as infrastructure, beyond the direct control or responsibility of the project. They may be only one facet of the infrastructure configured for each project, however.

The following is a list of various kinds of resources that will eventually be included in an ODM Handbook or Practitioner's Guide as formal assets.

ODM Candidate Domain Characterization Checklist
ODM Definitions for Workproduct MSLs:

languages for specifying Domain Interconnection Model, Domain Genealogy Model, Domain-Specific Lexicon, etc.

ODM Domain Information Sources Checklist
ODM Domain Model Types Checklist
ODM Domain Modeling Objectives Checklist
ODM Domain Selection Criteria List
ODM Guidelines for Domain Identification
ODM Guidelines for MSL Definition
ODM Model-Specific-Language (ODM MSL)
Domain Genealogy Model Language (DGML)
ODM Modeling Formalisms Checklist
ODM Organization Typology
ODM Organization Typology Model

D.6. ODM Terms

analogue
a relation between two or more exemplar artifacts that are both modeled as instances of a given category within the descriptive model. We say that two exemplars are analogues "with respect to" a given model category. We can also speak of "analogous exemplars" with a similar meaning

differentiated analogue
a feature with multiple specializations that can be linked to different exemplars.

architecture
Within the context of ODM, an architecture refers to a structure for inter-connecting a set of constituent components. The components can be workproducts from any point in the software life cycle. Thus, it is legitimate to speak of a **requirements architecture**, meaning a collection of requirements organized into an overall structure (e.g., a requirements document); a **test-case architecture**, denoting an arrangement of individual test cases according to some organizing principle appropriate to the testing methodology in use, etc. The conventional notion of an architecture familiar to most readers would be a **design** and/or **implementation architecture**, that is, an arrangement of software design workproducts or code modules according to an overall structuring principle.

The term architecture is also usually applied to entire systems. In the ODM context, where the*domain of focus* may frequently span only a sub-system scope of functionality, the term **architecture** can be applied in a relative sense to denote a structuring principle spanning the entire domain context. This means that the *architectural variants* produced during *prescriptive modeling* will be architectures for the entire domain scope, not necessarily architectures for the entire systems within which the domain functionality occurs. (However, the interface between the domain functionality and the surrounding system context will usually be addressed as part of an architecture variant.)

With the exception of the details mentioned above, ODM's notion of an architecture is compatible with most current work in the area of software architectures. These sources should be consulted for more detailed understanding of architecture issues, such as [Gar93].

artifact
Raw data about an exemplar system, such as a code component, a bug report, a requirements document. An **artifact** is distinguished from a *workproduct* in that it may be a workproduct of a*legacy system* not currently in use as part of any development process, or even of a system never deployed (aborted prototype, etc.) It is also distinguished in that an artifact is generally used in a context and for a use other than that for which it was first developed. For example, when a requirements document is used as an artifact, it is not being used to specify the behavior of a single system as would a "true" requirements document. Thus, the same document may be considered a workproduct in one context of use, an artifact in another.

An **artifact** is distinct from an*asset* in that it is not assumed to be under the control of a reuse/asset manager; that is, it generally is not incorporated into an *asset base*, and has not been specifically reengineered for reuse. In fact, the ODM process can be viewed largely as the systematic reengineering of **artifacts** into assets. An asset would then only be considered an artifact if it, in turn, was taken as data for study in a new context, e.g., a researcher studying the domain analysis process in order to improve it.

Finally, the term **artifact** denotes the fixing of information in actual documentary form (such as on-line textual information). Domain information, e.g., knowledge about the domain in an expert's head or expressed in a conversation, must be fixed into the form of some artifact, e.g., an interview report or transcription, before it can be dealt with concretely as domain information in the method. The conversion of processes to artifacts is one of the primary techniques in process/rationale capture, and is integral to discovering opportunities for generative techniques in reuse.

asset

A workproduct that has been developed or reengineered for reuse and placed under management within an asset base. Assets can be workproducts from any phase of the software engineering life cycle, including requirements, design, code components, test cases, etc. Assets can also be reengineered versions of informal engineering artifacts such as checklists, process descriptions, and guidelines.

Assets can include both static components, that are reused by being adapted and incorporated into deliverable systems, and tools and generators, that are reused by the application developer executing them as programs and generating tailored assets. These will in general be workproducts themselves, although cascaded invocation of generators is possible and, for more mature domains, may be increasingly used.

asset base (CFRP)

A collection of assets, usually organized according to some principle (in CFRP terms, a library model, in ODM terms, an **asset base architecture**), and supported via some tools/environment technology that can be considered the **asset base infrastructure**.

asset ensemble

Denotes the scope of the specifications produced at the end of the prescriptive modeling phase, before *asset implementation technology selection* is performed. An **asset ensemble** can be implemented with either a *component-based*, a *generator-based* or a *hybrid* approach.

The specification for an asset emsemble is based on feature profiles that are produced during prescriptive modeling. The rationale for use of the term is to emphasize that the specifications produced as part of the *prescriptive domain architecture* may be implemented by a single asset, a family of assets or some hybrid strategy. For example, if the implementation language will be Ada, the variability allocated to a given asset ensemble might be supported with several discrete modules, an Ada generic instantiated by the asset utilizer, or a more powerful part generator. The **asset ensemble specification** allocates the desired features and defers this more detailed design decision to the asset implementor.

The implication is that asset ensemble interfaces should remain reasonably stable across technology changes such as conversion from a part-based to a generator-based approach within one functional area of the domain. This encourages a mixture of technology approaches even within a single domain, and facilitiates ongoing evolution of the asset base.

audience

People that are expected to make use of a descriptive model.

The term "utilizers" (or the related "users") is avoided here, because it is associated with particular roles of the CFRP Reuse Engineering idiom, and might imply that models are created only for Asset Utilizers. In fact, models may be created with the creating Asset Creation team in mind as a potential audience, Asset Managers, or any of a number of distinct Utilizer groups.

The term "audience" is also used to imply the notion of modeling as "performance". In modeling one is planning and creating an experience for the audience, an experience that may foster cognitive transformations (learning, increased understanding, the suggestion of alternative perspectives, etc.). Since a model is not just an ordinary workproduct, but always has learning as well as productive objectives, those who interact with the model can usefully be thought of as the audience of a performance. (The use of this term, and this definition, is an example of a "modeling performance" based on the technique of applying an unexpected analogy domain.)

black-box reuse

A reuse strategy that disallows modification of tailoring of components by utilizers in unanticipated ways. Since there is rarely any way of enforcing this on utilizers, the policy in effect means that such tailoring will be unsupported and no feedback from such usage will be solicited.

business area (line of business)

In ODM discussion, this phrase is used to denote the overall business context in which the reuse program is being performed. In a typical commercial software concern, this line of business would probably be of wider scope than a particular product line or product family, but narrower in scope than the entire corporation. It generally refers to entire systems (e.g., avionics command centers) which, in ODM terms, would contain many functional areas usefully encapsulated as domains.

candidate

A potential member of a set of objects of inquiry, to be considered during a selection process. Usually candidates are documented in a Candidates List as a checkpoint before the selection process begins.

candidate exemplar

A system, application, or subsystem not yet been determined to be in or out of a given domain's scope.

components
Denotes static assets within an asset base (as opposed to generative assets that produce the reusable workproducts required through invocation on a specification). Often used in the context of **component-based** as opposed to *generator-based* implementation strategies.

context
Factors that make one instantiation of a process distinct from another. Also refers to information relevant to understanding why a given process was enacted in a particular way or why a particular workproduct or artifact was developed in a particular way, when this information is not incorporated into the workproduct itself.

context of use
The situation in which a given workproduct is expected to be used, or the context in which a given term is used. Reengineering for reuse can be thought of as a process whereby workproducts are reengineered to be suitable for a **broader** or **expanded context of use**. Alternatively, this transformation could be from an assumed **single context of use** to **multiple contexts of use.** A related phrase is *intended scope of applicability*. A scope of applicability consists of one or more specific contexts of use.

context-reflective
a qualifier for a workproduct indicating that its format results in explicit documentation of its own contextual assumptions. This is obviously only possible to a limited extent (e.g., if documenting context is considered a sub-process, then where is the context for context documentation documented, etc.) but a modest approach to this can yield very useful results and improve workproduct quality.

Can also be applied to a process if context-setting is an explicit substep of the process.

This is related to, but distinct from, process-reflective workproducts and processes. Most typically, context reflection comes near the start of the process, process reflection (including reflection on how well the formal or planned process was followed and what unanticipated processses took place) near the end of the process.

conventional software (system) engineering
customer
descriptive domain model
descriptive modeling
design for reuse
developer
domain
domain of interest (DOI)
domain of focus (DOF)
analogy domain

application context domain
[software] organization domain
a coherent body of knowledge about a class of software systems or system functionality, defined and shared by a communities of system developers and users within or across a set of stakeholder organizations.

sub-domain
domain analysis (DA)
domain analysis method
domain analysis representation
domain definition (process)
domain exemplar
domain information/data source
domain interconnection
domain relation
domain scoping (process)
domain-specific model
exemplar artifact
an artifact, drawn from any phase of the life cycle, from an exemplar system/subsystem for the domain.

exemplar [domain exemplar]
used to denote a system or subsystem defined as being a member of the domain. (Note the usefulness of the term "exemplar" to avoid repeating this awkward conditional phrase "system, subsystem, or distributed subset of functionality"!)

Usually the term refers to exemplars enumerated as part of the Domain Exemplar Set (DES). If, however, another potential exemplar is discovered that fits the criteria established by the Domain Definition Statement (DDS), and is not excluded as a counter-exemplar within the DES, it could be considered an exemplar.

exemplar system:
an exemplar for a domain whose scope includes entire application systems. . For "horizontal" domains, or domains that include slices of functionality within larger applications, the use of the term "exemplar system" denotes the entire system context within which the exemplar (subsystem) occurs. Since some domain information gathering activities will be planned at the level of overall systems (e.g., identifying experts and other informants for the system, obtaining documentation) this proves a useful term. When the intent is simply to denote the system, subsystem or embedded functionality pertaining to the domain, the less restrictive term "exemplar" by itself is preferable.

counter-exemplar
borderline exemplar
core exemplar

extensional definition
A definition expressed by enumerating example members of the set defined. Usually contrasted with *intensional definition*.

feature
A statement, usually in semi-formal text, about a capability or attribute associated with a system workproduct visible to some some set of stakeholders at some specific phase of the system life cycle.

feature (taxonomy | model)
A network of specializations of a feature

feature characterization
Determining what specialization within a feature model accurately describes the capabilities/attributes of an exemplar workproduct or an asset.

feature precedent exemplar
An exemplar in the Domain Exemplar Set that provides justification for inclusion of a feature variant in a domain feature model. This usually implies that and will also serve as a mapping of analogues within different systems.

feature profile
A set of characterizations of an exemplar workproduct or asset corresponding to a feature set.

feature restriction
feature set
A cluster of features related together for the purpose of characterizing exemplar systems. While there may be semantic constraints on instantiations of the feature set, no features in the set should be direct descendants of only other features in the set; in such a case, this feature should be allocated to the characterization process for one of the parent features, with the other parents reflected in this process as constraints where relevant

domain feature set
A feature set considered adequate to characterize exemplars in a domain for a specified set of stakeholder contexts.

feature variant
An immediate specialization of a feature within the feature model. An arbitrary descendant is a **feature specialization.**

feature: defining feature
Part of the domain definition.

feature: descriptive feature
A feature as developed in the Integrated Domain Model (assumed descriptive context), that is directly linked to the required categories within:

artifact model
conceptual model
contextual model
feature: differentiating feature
feature: infrastructure feature
feature: innovative feature
analogy domain
feature derived by analogy

descriptive feature
feature: Interpreted feature
A descriptive feature that has been placed into semantic relationship with other feature variants in a feature taxonomy. These taxonomies themselves may have defined semantic relationships...

architectural linkages
Like KAPTUR's recursive descent

could represent module interconnection architecture

but product-process interaction may mean that some choices at some level affect the module structure beneath

decision tree
Could represent decision tree, hence

process, historical oriented

needs to be mapped, by comparing to other historical processes, to a decontextualized process support or decision support model

feature: prescriptive feature
general business area
generators
infrastructure
infrastructure model
initiative
Within the context of a process model: the starting impulse for a series of data flows and activities that is not explained as a response to a prior stimulus.

Within the overall context of reuse programs, the term refers to: 1) the meta-process by which a reuse program is created within a given organization context; and 2) the first cycle or sequence of cycles through the Reuse Management processes for the program as a whole, during which the program serves the dual purpose of creating, managing and motivating reuse of asset bases, and improving the organization's capability of performing reuse-based software engineering.

The initiative can be seen as a special case of a transition function, relative to a model of capability or maturity levels, where an initiative moves the organization from stable performance at one level to a new level of performance. In this regard, a

special significance is given to the first initiative, where the organization moves from the informal reuse state to a managed state. This is not a simple sequence of activities to characterize, since different organizations will start from different levels even in terms of their informal reuse practice.

intensional definition

A definition that proposes a procedure for determining whether a given exemplar system, application or subsystem falls within a given domain.

The intensional definition includes a series of rules of inclusion and exclusion. Each consists of an attribute or feature by which a candidate exemplar can be characterized. Rules of inclusion and exclusion can use constraints (such as AND/OR conditions) to allow "short-circuit" evaluation of membership without necessarily evaluating all rules.

Some features within the intensional definition will be suffficiently described in the definition. Others will be the basis for differentiation of domain exemplars, based on more detailed characterizations according to specializations of the feature. Still other features may have complex enough variant possibilities that they are most properly dealt with as a sub-domain. Nevertheless, the presence of some variant of that feature may be a defining condition for the domain. E

Example: the domain of command centers may be defined to always include a DBMS, although the characteristics of that DBMS could be complex enough to warrant a separate DBMS domain model.

interest

a stakeholder has an interest in a domain, a system, or even a particular workproduct.

Related terms: domain of interest, focus, stakeholder

layered architecture
leverage
managed asset
model
model-driven

vs. information-driven; specialized by lexicon-driven

modeling

data-driven modeling
ontology-driven modeling

neutrality

initiative neutrality

term applied as a qualifier to a process model that allows variants where the

ontology
Organization Domain Modeling (ODM)

A domain analysis method, including a set of core concepts, enabling disciplines and key technologies, a process model including categories and specific tasks and

activities, a set of prescribed workproducts to be produced, and guidelines and heuristics for application of the method.

populate/inform
term that describes adding an object description into a model. This is distinct from adding a new specialization or a new relationship to the model itself. Often the addition of an object will necessitate an adjustment to the model structure itself, but this is not assumed in the term.

A possible point of ambiguity: we usually speak of the asset creators and/or asset managers populating the asset base. Populating a model is distinct, in that we are usually linking in a limited description of an asset or exemplar workproduct. A model need not be completely populated to be useful, and there may not be clear criteria for what "fully populated" means. (One interpretation; at least one object for every specialization within the model. Other, more or less rigorous criteria for "population completeness" could be proposed.

An alternative term is to "inform" the model, i.e., to place information into it in such a way that the model is potentially shaped and influenced by this information.

precedent
An instance within a domain exemplar artifact that supports the inclusion of a term or a concept in a lexicon or a domain model.

variant precedent
A set of two or more precedents that indicate instability in the meaning or semantics of a term or concept within the domain.

The variant will typically be drawn from distinct references within distinct artifacts, although it is possible to have a single artifact that seems to embody variant notions of a term's semantics.

The instability may be a result of implicit spanning of multiple domain contexts (where the meaning varies, in a "punning" sense, across these contexts) or of more fundamental instability in the domain terminology.

prescriptive domain model
prescriptive modeling
reference
A domain term or concept has *precedence* with respect to a given *exemplar set* if it is used within an *exemplar artifact*. The **reference** is the precise site of use where the term appears within the artifact. The notion of the reference allows the possibility that one artifact provides several references, even for the same term or concept.

related domain
A domain related to the *domain of focus* according to one of the domain interconnection or domain relation types that form part of the ODM Domain Interconnection Model.

repertoire

General term for a collection of strategies, techniques, heuristics, or structures that can be applied to particular design problems. Within the ODM context: ODM provides a **repertoire** of model innovation strategies for use in transforming descriptive models. Developers may have a **repertoire** of specific architectural idioms to apply during architecture evolution, or of system modeling notations to apply. No sequence or prioritization is implied among the repertoire elements. The term conveys the notion of a set of capacities that can be selectively applied in given situations. The term is borrowed from analogy with the performing arts (as is *audience*).

representative (domain representative exemplar)

One of the domain exemplars that have been selected for detailed study as a representative during descriptive modeling.

representative set (see Domain Representative Set)

resource

Information available in the performance of a process that is not essential to the form of the result nor necessarily transformed by the process itself (i.e., not an *input*), and is neither a polioy control or *constraint*, nor a performer of the process or automated tool support, i.e., a *mechanism*, in an SADT or IDEF sense. The classic **resource** would be a reusable software component or library of components utilized during the creation of a software workproduct. If the component (or generative tool) is drawn from a managed asset base, then it would be more specifically termed an *asset*. The term **resource**, however, applies not only to formal assets but to example workproducts adapted in an ad hoc way. (Some IDEF theorists may argue that this term properly belongs under IDEF categories.) This document refers to **resources** in each process description with the intent that some informal ODM materials could be included before they have the formal status of assets, and other materials could be included as well.

results/objectives

The changes or transformations that performance of a process effects in its context. This term includes broader phenomena than the more specific notion of *output*. Various kinds of results are obtained, in principle, from every process. **Product results** include *workproducts* created by the performance of the process. Some of these are formal *products* or *deliverables* of the process; these results are usually thought of as the primary motivation for performing the process. Besides these products, **interim workproducts** and **informal workproducts** may be produced.

scope of applicability

single-system scope of applicability

multiple-system scope of applicability

intended scope of applicability

The intended scope of applicability includes all contexts of use that were anticipated by the developer of a workproduct.

The intended scope of applicability can be inferred from ethnographic dialogue with the workproduct developer as an informant, or by inspection

of related workproducts such as requirements specifications for the workproduct.

The intended scope of applicability can differ from the scope of applicability for a workproduct. The actual scope of applicability may fall short of, or exceed, sometimes in subtle ways, the intended scope. The [actual[scope of applicability can be inferred from successful use of the workproduct in exemplar contexts, test results, or documentation of system behavior.

separately selectable
Adjective applied to a particular functional subset or subsystem of an overall system when it has been designed in such a way that it can be extracted as a stand-alone system or component. The design principle by which various subsystems of a system are considered for this property is **separate selectability**. An architecture which allows for various subsystems to be extracted in this way is a **separately selectable architecture.**

software engineering
In ODM discussion, software context assumed, so that engineering implies software engineering.

specialization domain
stakeholder context
a particular Create, Manage or Utilize context within the Domain Stakeholder Model

sub-model
term
an entry in a domain lexicon

term cluster
a set of terms interpreted by the modeler as synonymous, and clustered together under a standard canonic term as a representative.

The term cluster may also include acronyms or contextual abbreviations of compound terms as well as true synonyms. For example, "domain exemplar" and "exemplar" could be considered a cluster, where "domain exemplar" more fully qualifies the term "exemplar". Within a given domain context, a given term may be assigned a semantics that overlaps with the most commonly intended compound term; so, for example, it is not necessary in the ODM context to mention domain exemplars; but use of exemplar without the meaning of "domain exemplar" would need to be made explicit.

canonic term
one entry among possibly several synonym terms designated the standard version of the term for use in domain models and asset descriptions.

transformator
A more general term for a tool or mechanism that automatically transforms workproducts in some way. A *generator* is a transformator that, in general, takes high-level specifications and produces lower-level outputs, factoring in some content

decisions encoded into the generator itself. Transformators could also refer to tools that transform artifacts at roughly equivalent levels of abstraction. For example, a tool to convert subroutine calls to procedure calls, and vice versa, or a pretty-printer for a source language, might be more appropriately considered transformation rather than generation tools.

usability

In the context of ODM, the evaluation of whether a given feature or feature combination would serve some function within a given context of use.

usage

A type or category of use. The type of use to which a term or a workproduct is typically put. For example, the usage of the term "lexicon" in the DAPM DA methodology is different from the usage of the term "lexicon" in ODM. The usage for a requirements specification is an input to a design process. The usage of a code component is execution within an operational system.

use

The most general verb, meaning to take a workproduct as an input to a particular task or process. Tools or people (i.e., actors) **use** workproducts in performing tasks. Tools can be **used** in performing tasks. People cannot be used; they have roles that they perform for a given task, and they can be **users** of tools or workproducts, Multiple people involved in performing a task perform as a team. Workproducts used might or might not be formal assets.

utilize

To use a workproduct following a process that increases its value as a current or potential asset. If the workproduct is already an asset, the utilization process: respects the process of the asset base; contributes lessons learned about the workproduct back to the asset manager; and does not unintentionally violate the intended scope of applicability of the asset (although it may stress the asset for the purposes of innovation exploration). If the workproduct is not (yet) an asset, utilization should place the workproduct within the context of an asset base: either submitting the workproduct for reengineering into asset form before utilization, or reengineering the workproduct for leverage into the new context of use (i.e., using it as a resource) but submitting the reengineered form as a draft asset to the asset manager.

Example: A designer receives a requirements specification as input, and *uses* this requirements specification to develop her design. The design could not have been developed without the requirements specification. It is therefore an input to the process.

If the designer uses a prototype design as an example, modifying it as needed to suit the requirements specifications, the prototype design is used as a *resource*. The prototype design, helped, but did not enable, the designer to perform her task.

Use of the prototype design workproduct as a resource does not yet constitute *utilization*. If the prototype design was an asset in an asset base, and it is reused in accordance with the plans of the asset implementor and asset manager, it has been

utilized in the classic sense. Less typically, if the prototype design was not (yet) a managed asset, and the designer uses it in the context of an asset base, and through my use of it reengineer it to be closer to the requirements for an asset in that asset base, then it has been*utilized* in a less direct sense.

variant

immediate variant
descendant variant
leaf (most specific) variant
variant precedent (see xref)

workproduct

A textual or graphic document generated by human activity, existing in persistent (machine-processable?) form.

This term is written as a single word to emphasize that it is not a simple modifier of the concept of a product.

An artifact is a workproduct when it is being used for the purpose intended by its form and structure. For example, a requirements statement is intended to be used, directly, for validating customer needs and directing system designers. When used in this context, the requirement can be considered a workproduct (or an entity within a workproduct, i.e., a requirements statement within a requirements document).

When used as input to another process, e.g., domain analysis on requirements for the domain, the requirement is being used for a different purpose. In this context, it would be considered an artifact.

workproduct type

a category designated a differentiated class of workproducts

this category is part of the Domain Information Sources Model

instances of that model are in the Domain Information Sources Catalog

deliverable

a workproduct that is contracted for by the customer of a given system development effort. Workproducts include deliverables of a system development effort but are not confined to deliverables

non-deliverable workproduct

process-supported workproduct

a workproduct generated as part of an explicit process model followed in a project

informal workproduct

a workproduct not generated as part of an explicit process model.